**Problem Statement**

You are to implement a simple server using the C programming language and Linux unnamed pipes. The server will receive one-character requests from clients and send 4-byte replies. The client and driver code will be provided to you.

**Discussion**

The server is very simple and merely maintains a counter value for each client. The server responds to the following requests:

* '0' means to clear the client's counter and reply with the counter value.
* '+' means to increment the client's counter and reply with the counter value.
* '-' means to decrement the client's counter and reply with the counter value.
* '!' means the client is ending. There is no reply. Instead, the server should reduce the number of active clients.

While there are active clients, the server should repeat calls to select() to wait on activity on the different request pipes. You will need to use the current Linux version of select(), which is different from the select() described in Figure 3.7 in the textbook. You will need to use the FD\_ZERO and FD\_SET macros to initialize the set of read file descriptors for the request pipes prior to the main loop. You should send a copy of the set of file descriptors to select() on each iteration of the main loop. You will need to use the FD\_ISSET macro to test for activity on the copy of the set of file descriptors after select() returns with a positive value. When a client ends, you will need to use the FD\_CLR macro to remove the corresponding read file descriptor from the set.

The main loop is structured as follows:

while( active\_clients ){

prepare for the call to select()

retval = select( max\_fds, &query\_fds, NULL, NULL, &timeout );

respond appropriately

}

Note:

* retval == -1 means error; please print "error code returned from select()" and set active clients to 0
* retval == 0 means timeout; please print "timeout returned from select()" and set active clients to 0
* retval > 0 means there is request activity on one or more request pipe read file descriptors; please locate, read, and process the request(s)

You should use a timeout of 2 seconds as a parameter for the call to select().

To send a counter value back to a client, you must call write() for 4 bytes and cast the pointer to the integer counter value as a char pointer. The write() must be directed to the write file descriptor of the reply pipe that corresponds to the client making the request.

In the program 1 files subdirectory on Canvas you will find four source files: client\_server.h, clients.c server.c, driver.c. The server.c source code file is a skeleton, which you must complete. You are allowed but not required to define helper functions in server.c to make the code more readable. (E.g., I defined a helper function to encapsulate the counter update logic for my server.)

Please note that I may test your server with additional client types beyond those given in the clients.c source file. (For example, a client type that sends the '0' request.)

Turn in the completed source file named "server.c" using handin.cs.clemson.edu. Do not submit a compressed source file, and do not submit any other source files. Your code must run on the School of Computing servers (e.g., the titans) to be graded.

**Guidelines**

The code should be written totally by yourself.

You may discuss the project requirements and the concepts with me or with anyone in the class.

However, you should not send code to anyone or receive code from anyone, whether by email, printed listings, photos, visual display on a computer/laptop/cell-phone/etc. screen, or any other method of communication.

Do not post the assignment, or a request for help, or your code on any web sites.

The key idea is that you shouldn't short-circuit the learning process for others once you know the answer. (And you shouldn't burden anyone else with inappropriate requests for code or "answers" and thus short-circuit your own learning process.)

**Example Expected Output**

For the command line "./a.out dd", on one run you might see:

client/server test driver with 2 client processes  
client type d selected  
client type d selected  
client 0, type d, with file descriptors 3 4 5 6  
parent waits  
client 1, type d, with file descriptors 7 8 9 10  
server with 2 clients, which have file descriptors  
3 4 5 6  
7 8 9 10  
message from client 0 to server is -  
message from client 0 to server is -  
message from client 0 to server is -  
message from client 1 to server is -  
final reply from server to client 0 is -3  
message from client 0 to server is !  
message from client 1 to server is -  
message from client 1 to server is -  
final reply from server to client 1 is -3  
message from client 1 to server is !  
parent done

On another run, you might see:

client/server test driver with 2 client processes  
client type d selected  
client type d selected  
parent waits  
client 0, type d, with file descriptors 3 4 5 6  
client 1, type d, with file descriptors 7 8 9 10  
server with 2 clients, which have file descriptors  
3 4 5 6  
7 8 9 10  
message from client 0 to server is -  
message from client 1 to server is -  
message from client 1 to server is -  
message from client 0 to server is -  
message from client 1 to server is -  
final reply from server to client 1 is -3  
message from client 0 to server is -  
message from client 1 to server is !  
final reply from server to client 0 is -3  
message from client 0 to server is !  
parent done

For "./a.out b", you should see:

client/server test driver with 1 client processes  
client type b selected  
parent waits  
client 0, type b, with file descriptors 3 4 5 6  
server with 1 clients, which have file descriptors  
3 4 5 6  
timeout returned from select()  
parent done