PA6: The Server Moved Out!

Due Thursday 4/29/20 at 11:59pm, Total 100 pts

Introduction

In this PA, you are going to add a IPC class called NetworkRequestChannel to provide communication across the network over TCP/IP. Specifically, you allow the client-side end of a request channel to reside on one machine, and the server-side end of the channel on another machine. The communication over a request channel is to be provided by TCP connection(s). Since the communication API (not just the underlying functionality and features) through TCP is different from FIFO, you need to restructure the server.cpp and client.cpp as well.

First, you are to modify the server program from PA5 to handle incoming requests over network request channels instead of FIFO. The server must be able to handle multiple request channels from the same client (i.e., a single host, multiple threads) or from multiple processes on different machines. As a matter of fact, this is quite straightforward because the server side of a TCP connection does not care where the connection is being made from. You also must modify the client to send requests over network request channels.

The Assignment

You are to write a program (call it client.cpp) that consists of `p` patient threads, `w` worker threads, and file transfer using TCP/IP request channels. Then write your `server.cpp` so that multiple instances of the client program, either from the same or from different client machines, can connect to the server simultaneously. The client program is to be called in the following form for data requests:

```
./client -n <#reqs> -w <# workers> -b <bb size> -p <#patients>  -h
<host name / IP> -r <port no>
```

Or, like the following for file requests:

```
./client -n <#reqs> -w <# workers> -b <bb size> -f <filename>  -h
<host name> -r <port no>
```

The server is to be called in the following form: `./server -r <port no>`

Note that you the server no longer runs using `exec()` function from the client. Rather, it is run in the terminal in a different machine, or a different terminal in the same machine (i.e., TCP/IP also works as an IPC method). The server executable does not need the host name argument because it runs the service in the current host. It just needs to take the port number from the user that must occupy and reserve from the OS in that machine. The client on the other hand needs to know where the server is running (i.e., the domain name of the host or its IP address) and at which port number.

Implementation Note

Add a class called TCPRequestChannel which will replace `FIFORequestChannel` from PA5. The following should be the class declaration:

```
class TCPRequestChannel{
private:
     /* Since a TCP socket is full-duplex, we need only one.
     This is unlike FIFO that needed one read fd and another
      for write from each side  */
    int sockfd;

public:
     /* Constructor takes 2 arguments: hostname and port not
     If the host name is an empty string, set up the channel for
     the server side. If the name is non-empty, the constructor
     works for the client side. Both constructors prepare the
     sockfd  in the respective way so that it can works as a
server or client communication endpoint*/
   TCPRequestChannel (const string host_name, const string
port_no);

     /* This is used by the server to create a channel out of a
     newly accepted client socket. Note that an accepted client
     socket is ready for communication */

   TCPRequestChannel (int);

   /* destructor */
   ~TCPRequestChannel();

   int cread (void* msgbuf,  int buflen);

   int cwrite(void* msgbuf , int msglen);

    /* this is for adding the socket to the epoll watch list */
   int getfd();

};
```

You should provide definitions for the functions declared above in a `.cpp` file. In addition, there are some changes in the `server.cpp` and the `client.cpp` are necessary. For instance, the server should run an infinite loop to `accept()` incoming connections from the client and create a new thread with the accepted socket to process subsequent communication using the `handle_process_loop` function, which should stay mostly unchanged except for the input argument type (it should now be `TCPRequestChannel*` instead of `FIFORequestChannel*`).

The client on the other hand, can simply call the respective constructor `w` times to create all the channels. Unlike PA5, there is no need for sending a `NEWCHAN_MSG` to the server because each constructor gives a separate dedicated channel. For the same reason, the client does not need a separate control channel either. Finally, add a new entry in the makefile that compiles your `TCPRequestChannel` class.

What to Hand In

You are to hand in a .zip file that comprises the following files:

- All necessary code files and the makefile, even the ones that you did not modified from PA5
- Measure the performance of the PA6 under varying `w` for data requests and varying sizes of files - text and binary and compare with those from PA5. Present the results of these comparisons using graphs like previous PAs. In addition, verify correctness of file transfer by transferring a PDF file and `diff` ing the two
- Include a video demo with the report that follows the grading instructions and explains different important sections of your code, and answers the following questions:
    - How does TCP/IP method differ from FIFO in terms of speed?
    - What is the maximum number of connections you can create without changing the `ulimit` parameter? Is the number same as in PA5?
    - What happens to the point of diminishing return? Does it change from what you saw in PA5?

Getting Started

Use your PA5 as the starter code. That means, your code must have a single epoll thread. If your PA5 is not fully functional you have to fix it and for that you can seek help from the teaching staff. If you cannot salvage PA5 at all, you can use PA4 as the starter code, but that will incur 20% penalty.
For TCP/IP client-server, use the following example:
https://drive.google.com/drive/folders/1e5hQr7XXHfRkVGhR6WzIls7ZCHAgw9gV?usp=sharing