**Operating Systems**

**Final Project Part 5**

The final project consists of modifications to the producer-consumer system. For part 5, the final part, you will introduce a new type of client and make the requisite changes to the server.

You do not need to write a new client, but you can for a bonus. You may just use telnet to test this new feature. Because you (or I) might be using telnet (or other tools) to test this part, even though commands ending in “\r\n” are specified, your server should not check the endings for these command carefully and focus only on the content of the command itself. However, the server’s answers must be strictly delimited according to the protocol.

The new client is the ***status*** client. All its messages start with the string STATUS. It can be used to find out the current status of the server. It may connect to the server at any time, but it can only ask for one piece of information, and then its socket is closed.

A request of the status client is simple and fast, so a thread should not be used. As these are requests for statistics that may be changing rapidly, the exact value at the time of the call is not important.

In the table below, the integer value described inside curly brackets {} should be sent as an ascii string (not as an int value). The curly brackets **should not** be included in the response.

**PROTOCOL**

|  |  |  |
| --- | --- | --- |
| **Client Command** | **Server Response** | **Finally** |
| STATUS/CURRCLI\r\n | {*current clients in server*}\r\n | socket is closed |
| STATUS/CURRPROD\r\n | {*current producers in server*}\r\n | socket is closed |
| STATUS/CURRCONS\r\n | {*current consumers in server*}\r\n | socket is closed |
| STATUS/TOTPROD\r\n | {*total producers served*}\r\n | socket is closed |
| STATUS/TOTCONS\r\n | {*total consumers* *served*}\r\n | socket is closed |
| STATUS/REJMAX\r\n | {*total clients rejected for max*}\r\n | socket is closed |
| STATUS/REJSLOW\r\n | {*total clients rejected for slow*}\r\n | socket is closed |
| STATUS/REJPROD\r\n | {*total producers rejected for max*}\r\n | socket is closed |
| STATUS/REJCONS\r\n | {*total consumers rejected for max*}\r\n | socket is closed |

* Current clients – total clients currently being served
* Current producers – clients who have identified themselves as producers and are currently being served
* Current producers – clients who have identified themselves as consumers and are currently being served
* Total producers served – who reached the point of placing their item in the item buffer
* Total consumers served – who reached the point of removing an item from the item buffer
* Total clients rejected for exceeding client max
* Total clients rejected for taking too long to identify themselves
* Total producers rejected for exceeding producer max
* Total consumers rejected for exceeding consumer max

Final Project Requirements

Each part of the final project is mostly independent (except that part 5 needs the 5 counters listed in part 4). It will be much better (grade-wise) to leave out one part than do multiple parts poorly or not completely.

Every student is required to have an online meeting with me to go over the final project as a whole by May 8, 2020. In this meeting I may run tests, or I may ask you to demo to me, and I may ask you to explain sections of your code.

You must write a report to go with your final project. For each of the project parts you completed, discuss the choices you made to solve the part, and any challenges you faced.

Some parts, for example part 4’s requirement to reject slow clients, leave a lot of implementation decisions in your hands, whereas others, like part 2, do not. Clearly you will have more to write for some parts than others.

If you have been doing online meetings with me discussing implementation decisions and challenges all along, your report can be very brief and will contribute very little to your grade. On the other hand, if you choose to have just the final meeting, then the report will be an important part of documenting your work.

Optional client program(s)

1. You may write a single status value client, where its name its name is status, and its arguments are as follows:

status [host] port value

* where value is a string in the set of strings that follow STATUS/ in the protocol above, meaning the value argument should not contain “STATUS/”.

The client should print a descriptive message about the value requested in response, and exit.

1. You may write a multiple status value client, which stays open allowing the user to enter requests for status values until the user types “Q” to quit. The client would start with:

status [host] port

Then it would display some sort of menu to the user, for example:

Enter the status value you want to get or q to quit:

You create any menu style you want and display the results in a descriptive way.