**Operating Systems**

**Homework – Producer/Consumer Server and Clients**

In this homework, you will create a server that manages an example of the producer-consumer problem. You have a working solution to the producer-consumer problem in moodle.

In this system, producers and consumers are clients of the server, and you will write them as well.

Some definitions:

CR – ascii code decimal 13, ‘\r’

LF – ascii code decimal 10, ‘\n’

The server keeps a fixed-sized buffer to store the items:

typedef item\_struct {

int size;

char \*letters;

}ITEM;

Each producer client contacts the server and sends it an ITEM, but it must wait until there is room in the buffer to place an item.

Each consumer client contacts the server and to get an ITEM sent to it, but it must wait until one is available.

The server’s job is to manage the producers and consumers using the buffer, putting new ITEMs from producers into the buffer and handing them out to consumers who request them.

The series of messages goes as follows:

|  |  |
| --- | --- |
| Producer client connects to server, sends | PRODUCE\r\n |
| When a space is available in the ITEM buffer, server sends | GO\r\n |
| Producer sends a 4-byte int indicating the number of chars to follow | *letter-count* |
| Server instantiates the ITEM |  |
| Producer sends the characters | *stream of chars* |
| After the item has been added to the buffer, server sends, then closes | DONE\r\n |
|  |  |
| Consumer client connects to the server, sends | CONSUME\r\n |
| When an ITEM becomes available, server sends a 4-byte int indicating the number of chars to follow | *letter-count* |
| Server sends the characters, then closes the connection | *stream of chars* |
| Client writes the chars into a file and then closes it |  |

The maximum number of simultaneous clients that can be handled by the server is 512.

The character strings in the ITEMs can be any size and must be allocated dynamically.

The ITEM buffer size should be fixed and you will try multiple sizes, passed in on the command line.

The QLEN of the server socket should be 5.

Producer client:

Produces a random-sized array of individually random printable characters. Then it contacts the server and sends the token, PRODUCE, followed by a CRLF. After the server replies GO followed by CRLF, the producer sends a 4 byte int containing the number of characters that will be sent, and immediately streams its array of chars over to the server and then closes the connection and exits.

Consumer client:

It contacts the server and sends the token, CONSUME, followed by CRLF. It then awaits a 4 byte int containing the number of characters to follow, which is immediately followed by a stream of chars from the server, which then closes the connection. The client then writes these characters into a file named by its pid, pid.txt, and then exits. The file handling should be done using the system call level interface.

The clients/servers may write informational messages to the console, but please do not leave debugging that writes the entire stream of characters to the console. Absolutely no extra characters should be sent over the network. Your clients and servers will be tested by our own clients and servers, so they must comply exactly with the specification.

Write one main consumer program that launches N individual, independent consumers (threads), and one main producer program that launches N individual, independent producers (threads).

consumer host port N

producer host port N

cpserver [port] bufsize

**Error handling:**

If more clients arrive at the server than it can handle, the server may just close the connection to those clients. For example, if the server is already handling 512 clients (there are 512 active threads), then when the 513th client arrives, the server may immediately close its connection, and so on, until one of the active threads finishes. Notice that keeping track of this will likely create a critical section in your code.

Simple networking errors, such as the socket being closed unexpectedly, should just result in the handling thread exiting. The whole process should only exit due to serious errors (error from pthread\_create, error from accept, etc.), or errors with the command-line arguments.