Operating Systems. Homework #4.

Submission - June 18, 23:55.

In this exercise, we implement Printable Characters Counter (similar to homework #1) using client – server architecture. We develop two executables – pcc_client and pcc_server. The overall flow is the following. Client sends some byte stream to Server through a TCP connection, Server counts the number of printable characters in this stream and returns the number to Client. In addition, Server maintains a data structure that accumulates the statistics of all printable characters that were processed in this run. The content of the data stream is extracted from /dev/urandom.

Statistics data structure

This is a data structure that registers how many bytes of specific value were observed. Something like - "we saw 11 'a's, 43 'b's, 125 'c's...". It can be an array of integers, where the index is the corrected ASCII value of a byte (-32), and the values of the array are the counters. Any alternative implementation (but without disk I/O involved) is allowed.

Client (pcc_client)

Command line argument:
LEN - the length (in bytes) of the stream to process. Assume an integer number.

The flow:
1. Open a socket to Server on your local machine, port 2233.
2. Open /dev/urandom for reading.
3. Transfer (write) LEN bytes from /dev/urandom to Server through the socket.
4. Get (read) the result from Server through the socket.
5. Print the result.
6. Close descriptors.
7. Quit.

Server

The flow:
1. Register signal handler for SIGINT signal. (Ctrl-C pressed)
2. Create and initialize statistics global data structure (GLOB STATS).
3. Initialize the global counter of total bytes read. (An integer)
4. Create and initialize data mutex.
5. Listen to port 2233 in an infinite loop.
6. Upon connection accepted - start a Client Processor thread, continue listening.

Client Processor Thread:
Argument:
Connected client socket descriptor.
The flow:
1. Create a statistics local data structure.
2. Read the content from the socket. For every byte:
   a. Increment the number of bytes read.
   b. Decide whether it is printable or not.
   c. If it is, then update the local statistics.
3. Send the number of the printable bytes back to the client.
4. Close the connection.
5. Acquire (lock) mutex.
6. Add local statistics to GLOB STATS.
7. Add the number of bytes read by this thread to the global counter.
8. Release (unlock) mutex.
9. Exit thread.

Upon SIGINT signal:
1. Stop listening the port. Close the socket.
2. Wait for all running threads to finish.
3. Print the global bytes counter.
4. Print the global statistics.
5. Exit process.

Submission

Submit a zip archive named ex4_012345678.zip, where 012345678 is your ID. The archive contains 2 files `rcc_client.c`, and `rcc_server.c`. *Mac users! Don't submit hidden folders!*

Hints

1. While testing/debugging, Client can read a short text file with predefined content instead of /dev/urandom.
2. You can use NetCat utility (**`nc`**) to simulate Server or Client.
3. You can use **`ngrep`** utility to monitor the traffic on specific port. Pay attention that root permissions are required to run it. Use sudo.

Example:

Console 1

```
$nc -l 2233  #Server listens on
             #port 2233
Hello, world #Server gets the
             #string and prints
             #it.
```

Console 2

```
$nc 127.0.0.1 2233  #Client connects
Hello, world        #to port 2233, and
                    #sends the string
```

Console 3

```
$sudo ngrep -d any port 2233                #Start intercepting IP traffic
[sudo] password for eug:                    #on any device, port 2233
interface: any
filter: (ip or ip6) and ( port 2233 )
####                                        #Server accepted connection
T 127.0.0.1:41104 -> 127.0.0.1:2233 [AP]    #Transfer started
  Hello, world.                             #Dumping the data
####                                        #Connection closed
```