Machine Problem 8: The Data Server Moved Out!

Due 5/5/17 at 11:59pm, Total 100 pts

NOTE: Due to grade turn-in restriction for Spring Semester, the dead-drop late policy for MP8 is Sunday May 7, 11:59pm

Introduction

In this machine problem, we improve the request channel to provide communication across the network, which means each worker thread will communicate with the dataserver through the network channel that you are going to make. We 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 network request channel is to be provided by a single TCP connection.

To establish request channels over the network, the interface of the request channel must be modified somewhat. Below is a definition of the NetworkRequestChannel class that supports request channels across machine boundaries:

```
/* Creates a CLIENT-SIDE local copy of the channel. The channel is connected
to the given port number at the given server host. THIS CONSTRUCTOR IS CALLED
BY THE CLIENT. */
NetworkRequestChannel(const string _server_host_name, const unsigned short
_port_no);

/* Creates a SERVER-SIDE local copy of the channel that is accepting
connections at the given port number. NOTE that multiple clients can be
connected to the same server-side end of the request channel. Whenever a new
connection comes in, it is accepted by the server, and the given connection
handler is invoked. The parameter to the connection handler is the file
descriptor of the slave socket returned by the accept call. NOTE that the
connection handler does not want to deal with closing the socket. You will
have to close the socket once the connection handler is done. */
NetworkRequestChannel(const unsigned short _port_no, void *
(*connection_handler) (int *));

/* Destructor of the local copy of the channel. */
~NetworkRequestChannel();

/* Send a string over the channel and wait for a reply. */
string send_request(string _request);

/* Blocking read of data from the channel. Returns a string of characters
read from the channel. Returns NULL if read failed. */
string cread();

/* Write the data to the channel. The function returns the number of
characters written to the channel. */
int cwrite(string _msg);
```

You are to modify the data server program from MP7 to handle incoming requests over network request channels instead of request channels. The data server must be able to handle multiple request channels, either from the same client or from different clients, possibly on different

machines. You also have to modify the client from MP7 to send requests over network request channels.

The Assignment

You are to write a program (call it client.cpp) that consist of a number of request threads, one for each person, a number of worker threads, and a number of statistics threads, one for each person. The number of persons is fixed to 3 in this MP (Joe Smith, Jane Smith, and John Doe). The number of data requests per person and the number of worker threads are to be passed as arguments to the invocation of the client program. As explained earlier, the request threads generate the requests and deposit them into a bounded buffer. The size of this buffer is passed as an argument to the client program.

Design your dataserver (feel free to change it in the manner to best accommodate it to your design) so that multiple instances of the client program, either from the same or from different client machines, can connect to the dataserver 'simultaneously'. The client program is to be called in the following form, you can use your local IP address "127.0.0.1" as the IP in the program. You can use others if they work as well. Choose a big port number over 10000 as smaller ones may be reserved for well-known ports.

```
client  -n <number of data requests per person>
        -b <size of bounded buffer in requests>
        -w <number of request channels>
        -h <name of server host>
        -p <port number of server host>
```

The data server is to be called in the following form:

```
dataserver  -p <port number for data server>
            -b <backlog of the server socket>
```

Where backlog is the parameter that is provided to the listen system call executed by the server.

Report

1. Report a brief performance evaluation of your code. If there is a difference in performance from MP7, explain it. If the performance appears to have worsened, can it be justified as a necessary trade-off?
2. Measure the performance of the system with varying number of backlog on the server side and varying number of clients. Use the average response time from the server to the client as the "y-axis" value and number of client requests as the "x-axis" value. Plot multiple curves with varying backlog parameter. To make the server busy with the incoming requests, you may want to use large number of client requests to make multiple clients waiting in the "queue". State your assumption on timing start-stop instance.
3. Describe the platform that your data was gathered on and the operating system it was running. A simple description like "a Raspberry PI model B running Raspbian OS," or "the CSE Linux server," is sufficient. (Think of this as free points)

What to Hand In

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

- Your implementation of the Network Request Channel, to be submitted in two files: `NetworkRequestChannel.h` and `NetworkRequestChannel.cpp`
- The updated `client.cpp`
- The updated `dataserver.cpp`
- Any other file that is needed to compile and test your solution, including the updated `Makefile`
- Report File

Rubric:

You must make sure that the connection between server and clients is created successfully and multiple clients can connect to the server. That's the basic meaning of correctness in this project. If your program crashed or does not compile, it means it's wrong.

Code Portion: 80 points. Correctness is based on manual grading.

- Connection Created: 10 points
- Multiple Clients Accepted: 15 points
- Statistic Results are correct: 45 points
- Global Variables: 10 points (1 point off for each global variable usage)

Report Portion: 20 points. Do it by following the above instructions.

- Plots: 10 points
- Code Evaluation: 5 points
- Performance Analysis: 5 points