Machine Problem 2: (Some) Working with Processes

Introduction

In this machine problem, we set the stage for a high-performance data query and processing system. In a first step you are to implement a *client* program that first loads a *data server* program into memory and then requests data from that server.

The client program (to be implemented as program `client.C` is to fork off a process, which is supposed to load the data server program (the source code is provided in file `dataserver.C`. The client then issues requests to the server through what we will be calling *request channels*. Request channels are a simple inter-process communication mechanism, and they are provided by the class `RequestChannel`. Request channels provide the following interface:

```
RequestChannel(const string _name, const Side _side);
/* Creates a "local copy" of the channel specified by the given name.
   If the channel does not exist, the associated IPC mechanisms are
   created. If the channel already exists, this object is associated
   with the channel. The channel has two ends, conveniently called
   "SERVER_SIDE" and "CLIENT_SIDE". If two processes connect through
   a channel, one has to connect on the server side and the other on
   the client side. Otherwise the results are unpredictable. */

~RequestChannel();
/* Destructor of the local copy of the channel.
   By default, the Server Side deletes any IPC mechanisms associated
   with the channel. */

string send_request(string _request);
/* Send a string over the channel and wait for a reply. This function
   returns the reply to the caller. */

string cread();
/* Blocking read of data from the channel. Returns a string of characters
   read from the channel. Returns NULL if read failed.
   THIS FUNCTION IS LOW-LEVEL AND IS TYPICALLY NOT USED BY THE CLIENT. */

int cwrite(string msg);
/* Write the data to the channel. The function returns the number of
   characters written to the channel.
   THIS FUNCTION IS LOW-LEVEL AND IS TYPICALLY NOT USED BY THE CLIENT. */
```

You will be given the source code of the data server (in file `dataserver.C`) to compile and then to execute as part of your program (i.e. in a separate process). This program handles three types of incoming requests:

`hello` : The data server will respond with `hello to you too`.

`data <name of item>` : The data server will respond with data about the given item.

`quit` : The data server will respond with `bye` and terminate. All IPC mechanisms associated with established channels will be cleaned up.

The Assignment

You are to write a *program* (call it `client.C`) that first forks off a process, then loads the provided data server, and finally sends a series of requests to the data server. Before terminating, the client sends a `quit` request and waits for the `bye` response from the server before terminating. You are also to write a *report* that briefly compares the overhead of sending a request to a separate process compared to handling the request in a local function.

What to Hand In

- You are to hand in one file ZIP file, with called `Submission.zip`, which contains a directory called `Submission`. This directory contains all needed files for the TA to compile and run your program. This directory should contain at least the following files: `client.C`, `dataserver.C`, `reqchannel.C`, `reqchannel.H`, and `makefile`. The expectation is that the TA, after typing `make`, will get a fully operational program called `client`, which then can be tested.
- Submit a report, called `report.pdf`, which you include as part of the directory `Submission`. In this report you present a brief performance evaluation of your implementation of the client.
- Measure the invocation delay of a request (i.e. the time between the invocation of a request until the response comes back.) Compare that with the time to submit the same request string to a *function* that takes a request and returns a reply (as compared to a separate process that does the same). Submit a report that compares the two.