ERRS: Homework 4
Scalability

For this homework, you will be writing a server for a "bank". The focus of this homework is scalability, so the feature set for the "bank" is pretty minimal. In particular, you will not have a web interface, nor any notion of users or authentication—your bank will accept XML documents over network sockets (listening on port 12345) which describe account creations, transfers, balance inquiries, and/or queries for lists of transactions. The details of these are explained later.

There are four major steps for this project:

1. First, you are going to make a single-threaded version of this software, without focusing on scalability. Your Makefile should compile this version of the software into a binary which is named `bank-st`. You should functionally test this software extensively before proceeding. As part of this step, you should also make testing infrastructure for functional correctness. Namely, you should create a script or program called `func-test`, which should take exactly one argument (the host to connect to and test). This infrastructure should run a suite of tests (of your devising) and ensure the correct behavior of the bank server. Your testing infrastructure MAY NOT reject ANY behavior which is correct according to the spec below. In particular, you MAY NOT enforce implementation-specific choices that you have made (such as the particular information in an error message, the ordering of items in the response XML, etc).

2. Second, you should make load testing infrastructure. This may be a combination of programs, scripts, and data files. You (or your TA or I) should be able to run this all with a top-level script (or program) called `load-test`, which should take exactly one argument which is the hostname to send requests to (so `./load-test localhost` would load test the localhost, for example).

3. Third, you should make a scalable version of your bank server. Your Makefile should compile this version into a binary which is named `bank-mt`. You should still be able to compile `bank-st` after you are finished with this step (you might use conditional compilation directives, a different file with main, or any other approach you want to support both versions at once). Your goal is to make this version as scalable as possible. You may wish to update your `func-test` to test multi-threaded behavior. However, it must accept any valid ordering of in-flight transactions as correct.

4. Finally, you should write a report analyzing the performance data you have gathered, comparing your `bank-st` to your `bank-mt`. Graphs are strongly encouraged. If you have data which shows a particularlly slow or problematic behavior, you should include and explain it.

Note that your TA should be able to run any group's test infrastructure with any group's code. You are welcome to have another group run their test suite against your server, however, you should not share any code between groups. If you want to do this, you should tell the other group the hostname where you are running your server, and they should send you back only the output of their testing suite. If their infrastructure reports an error, you are welcome to tell them that you believe your code was correct, or agree that it found a bug, but you cannot help them debug their test infrastrucutre, nor can they help you debug your code.

The Spec

When your server accepts a request, it will first read 8 bytes, which is are a network byte order, unsigned 64-bit integer. This integer specifies the nubmer of bytes of the XML document, which comes immediately after the integer. This input XML document has the following requirements:

- The root element of the xml document MUST be `<transactions>`.
- The root element may optionally have one attribute: `reset="true"`. If this attribute is present (and its value is true), then your server should reset to a clean state before processing any transactions in this file (no accounts exist, no transfers have been performed).
- Except for the root element (which MUST be transactions), you SHOULD ignore any elements or attributes which are notexplicitly described by this specification.
- Inside the root element, there may exist zero or more `<create>`, `<transfer>`, `<balance>` and/or `<query>` tags.
- Each tag inside of the root element SHOULD contain an attribute called `ref`, whose value will be used to identify the corresponding response.

  create The `<create>` tag requests creation of an account. It MUST contain one `<account>` tag, and MAY contain one `<balance>` tag. The account tag MUST contain one textual element, which MUST be a valid 64-bit unsigned decimal integer value, which is the account number to create. If the `<balance>` tag is present, it MUST contain one textual element which MUST be a valid floating point number, which is the initial balance of the account. If no balance tag is present, the initial account balance is 0.

  transfer The `<transfer>` tag requests transfer of funds between existing accounts. It MUST contain exactly one of each `<to>`, `<from>`, and `<amount>`. It MAY contain any number of `<tag>`s.

  - The `<to>` and `<from>` tags must contain a textual element which specifies the account number to transfer money to and from respectively. If these accounts do not already exist, your server MUST respond with an error.
  - The `<amount>` tag must contain a valid floating point number, which specifies the amount of money to transfer. If the `<from>` account does not have sufficient funds (or the `<to>` account does not have sufficient funds if amount is negative), you MUST respond with an error.

  - Each `<tag>` contains one textual element, containing arbitrary text. The tags contain extra information about the transfer, and the server MUST record all associated tags.

balance The `<balance>` tag indicates a balance inquiry. It contains one `<account>` tag inside it. The `<account>` tag contains one textual element with the account number to check the balance of. If this account number is not valid, your server MUST respond with an error.

query The `<query>` tag requests a list of the transfers meeting some criteria (as specified inside of the `<query>` tag). The tags inside the query tag (query-specification tags) are a combination of `<or>`, `<and>`, `<not>`, `<equals>`, `<less>`, `<greater>`, and/or `<tag>`. These come in three categories:

Logical Operators : The `and>+, \verb+`or¿+ and `<not>` tags represent logical operations (and/or/not, respectively) of the query specificiations contained inside of them. Whenever multiple query specifications are present that are not inside an `<or>` or `<and>` (either directly inside a `<query>` or inside of a `<not>`), they are implicitly considered to be inside of a `<and>`. An empty `<and>` is always true (and thus matches any query), while an empty `<or>` is never true (and thus matches no queries). An empty `<not>` implicitly contains an empty `<and>`, so is always false.

Relational Operators . The relational operators, `<equals>`, `<less>`, and `<greater>` Each of these tags has exactly one attribute: `from`, `to`, or `amount`. The value of these attributes is an account number (for from/to), or a money amount (for amount). A transaction satisifes the criteria of this specification if its from/to/amount is equal/less/greater than the value of the attribute of this tag.

Tag The `<tag>` tag has one attribute, `info`, whose value specifies the tag to search for. This criteria matches any transfer which was tagged with text that is the same as the value of the `info` attribute.

Your server should respond with to the request with an 8-byte, network byte-ordered unsigned integer, which specifies the number of bytes of the XML response immediately following it. The server should then send an XML document with the following specificiations:

- If the input document is unparseable, the response MUST contain a single`<error>` tag, containing a description of the problem.

- If the input document was parsed (even if errors occured in specific transactions), the response MUST have a root element of `<results>`.

- The `<results>` tag MUST contain one of either a successful response tag or an error tag for each transaction (create, transfer, balance, query) which was contained in the input document. If the original transaction had a `ref` attribute, the response MUST include a `ref` attribute with the same value. If the original transaction did not include a `ref` attribute, the response MAY (but does not need to) include a `ref` with any text value that it wants, so long as that text was not used as the value of a `ref` of any input transaction.

- An error response to any transaction is a `<error>` tag with a textual element describing the error.
- A successful response to a `<create>` MUST be `<created>` tag, with no other information.
- A successful response to a `<transfer>` MUST be a `<transferred>` tag, with no other information.
- A succesful response to a `<balance>` MUST be a `<balance>` tag containing one textual element, which is is the balance of the requested account.
- A successful response to a `<query>` MUST be a `<results>` tag containing zero or more `<transfer>` tags. There MUST be exactly one `<transfer>` tag per transfer which matched the specifications of the query.
  - Each `<transfer>` tag MUST contain exactly ONE each of `<from>`, `<to>`, and `<amount>`. Each of these gives the from account id, to account id, and amount of the transaction, as a textual element within the tag.
  - If the matching transaction had any tags assocaited with it, then the `<transfer>` tag must also contain one `<tag>` per each tag associated with the transaction. The `<tag>` tag should contain the text of the tag (as it was in the original transfer request).

An example input file is:

```
<?xml version="1.0" encoding="UTF-8"?>
<transactions reset="true">
  <create ref="c1">
    <account>1234</account>
    <balance>500</balance>
  </create>
  <create ref="c2">
    <account>5678</account>
  </create>
    <create ref="c3">
    <account>1000</account>
    <balance>500000</balance>
  </create>
  <create ref="c4">
    <account>1001</account>
    <balance>5000000</balance>
  </create>
  <transfer ref="1">
    <to>1234</to>
    <from>1000</from>
    <amount>9568.34</amount>
    <tag>paycheck</tag>
    <tag>monthly</tag>
  </transfer>
  <transfer ref="2">
    <from>1234</from>
    <to>1001</to>
    <amount>100.34</amount>
    <tag>food</tag>
  </transfer>
  <transfer ref="3">
    <from>1234</from>
    <to>5678</to>
    <amount>345.67</amount>
    <tag>saving</tag>
  </transfer>
  <balance ref="xyz">
    <account>1234</account>
  </balance>
  <query ref="4">
    <or>
      <equals from="1234"/>
      <equals to="5678"/>
    </or>
    <greater amount="100"/>
  </query>
</transactions>
```

For this input, you might receive the following response:

```
<?xml version="1.0" encoding="UTF-8" ?>
<results>
  <success ref="c1">created</success>
  <success ref="c2">created</success>
  <success ref="c3">created</success>
  <success ref="c4">created</success>
  <success ref="1">transferred</success>
  <success ref="2">transferred</success>
  <success ref="3">transferred</success>
  <success ref="xyz">9622.33</success>
  <results ref="4">
    <transfer>
      <from>1234</from>
      <to>1001</to>
      <amount>100.34</amount>
      <tags><tag>food</tag></tags>
    </transfer>
    <transfer>
      <from>1234</from>
      <to>5678</to>
      <amount>345.67</amount>
    <tags><tag>saving</tag></tags></transfer>
  </results>
</results>
```

Note that if you removed the `reset="true"` and gave the bank the same input, you would receive different output. The accounts you have requested to create already exist (so those requests would produce errors). The transfers would still succeed. However, the account balance inquiry, and transaction search would produce different results (since there is now more money in the account, and more transactions matching the criteria). The results you might obtain are shown on the next page:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<results>
  <error ref="c1">Already exists </error>
  <error ref="c2">Already exists </error>
  <error ref="c3">Already exists </error>
  <error ref="c4">Already exists </error>
  <success ref="1">transferred </success>
  <success ref="2">transferred</success>
  <success ref="3">transferred</success>
  <success ref="xyz">18744.66</success>
  <results ref="4">
    <transfer>
      <from>1234</from>
      <to>1001</to>
      <amount>100.34</amount>
      <tags><tag>food</tag></tags>
    </transfer>
    <transfer>
      <from>1234</from>
      <to>5678</to>
      <amount>345.67</amount>
      <tags><tag>saving</tag></tags>
    </transfer>
    <transfer>
      <from>1234</from>
      <to>1001</to>
      <amount>100.34</amount>
      <tags><tag>food</tag></tags>
    </transfer>
    <transfer>
      <from>1234</from>
      <to>5678</to>
      <amount>345.67</amount>
      <tags><tag>saving</tag></tags>
    </transfer>
  </results>
</results>
```