Deliverable: Assignment #3
Title: CS436: Distributed Computer Systems
Offering: Winter 2011

WWW: http://bit.ly/uw_446-11 [The web page will be periodically updated]
Twitter: @cs436 (http://twitter.com/cs436) [Important updates broadcast here]

Lectures: Tuesday & Thursday 0830 - 0950 MC 4064

Instructor: Dr. Reid Holmes; DC 3351. Office hours by appointment. rth.cs436@gmail.com
TAs: Ali Abedi; DC 3549. Office hours by appointment. a2abedi@cs.uwaterloo.ca

Due: 0830 on Mar 31 via email to rth.cs436@gmail.com

Overview

In this assignment, you are asked to implement a simulated distributed hierarchical naming system. The system is implemented as a number of cooperating name servers, each of which represents a domain in the overall name space, similar to the Internet Domain Name System (DNS). The naming system resolves names to a corresponding integer numbers. Your task is to implement an appropriate name server and a client that can be used to access the naming system. The server must be capable of performing both iterative and recursive name lookups and further, client and server must be able to handle both corresponding reply types. To keep things simple, all name server and client processes are executed on the same machine.

Details

Each name server is started with a single argument:

```
server <filename>
```

where `<filename>` denotes the name of a configuration file. The syntax of the configuration file is specified below.

The client is invoked with a single argument:

```
client <name>
```

where `<name>` denotes the absolute name that is being looked up in the naming system. An absolute name contains a variable number of edge labels, represented by alpha-numerical strings, separated by a dot. In contrast to the typical DNS notation, labels in a name are ordered from higher to lower layers in the naming hierarchy. Consequently, absolute names start with a dot whereas relative names start with a label.

The server configuration file describes the domain that is being represented by a server, as well as the configuration of the name resolution process. It is an ASCII file that may contain the following types of attributes (an example is given below) and attribute fields:

```
DOMAIN      <name>      <transport id>
NS          <label>     <transport id>    <flags>
ID          <label>     <entity>
```

Each attribute entry in the configuration begins with one of the keywords `DOMAIN`, `NS` or `ID` and is followed by the attribute fields, each of which is coded as a string (containing alphanumeric characters or dots) and separated by whitespace. Each attribute entry must be written in a separate row of the configuration file.

Attribute fields are:

<name> denoting an absolute name as represented in the naming system,
<label> denoting an edge label,
<transport id> specifying a transport identifier (see below),
<flags> being either I or R for (I)terative or (R)ecursive name resolution, and
<entity> denoting the integer number of the abstract entity corresponding to a name.

Only a single DOMAIN attribute must be given in any configuration file. The <name> field contains the absolute name prefix of the domain represented by the server. The <transport id> field contains the local transport identifier for this server.

There may be an arbitrary number of NS attributes present in the configuration file. These entries refer to other name servers that can be contacted to obtain information about subdomains. The <label> field contains the edge label of each subdomain while the <transport id> field contains the transport identifier of the respective name server.

There may be an arbitrary number of ID attributes present in the configuration file. These entries provide mappings from an edge label <label> to an entity number <entity>.

System Structure

The software system consists of a server program and a client program. Multiple processes of the server program are started with different configuration files to represent a name space. Each invocation of the client program performs a single lookup operation and finishes upon reception of a response by printing the entity's id or adequately reporting about a potential lookup failure. You must implement the communication between the client and servers, as well as between servers using a suitable communication protocol on top of TCP or UDP and with the help of indirect transport addressing as described below.

A name server reacts to incoming requests and after carrying out basic checks, first performs an internal lookup on the requested name. If the entry found specifies an entity number (ID), the result is transmitted back to the requester. Otherwise (NS), subsequent actions depend on the flags of the name server entry. If no matching entry can be found, an error code is returned. Each lookup request must generate a response.

The communication protocol must support both iterative and recursive lookups. The details of the communication protocol are left for you to design. If a name server is configured to perform a recursive lookup for a name, it responds with the entity number found for the name. In case of an iterative lookup, a name server responds by referring the requester to the next name server, if necessary. Both server and client programs must be able to accept both types of responses and continue the name resolution in case of an iterative lookup. The server program should write information into a per-process log file which illustrates execution of the name resolution. Explain the format of your logging output in the README file. The server logfile should be named nameServerLog<name>.<number> where <name> is the name of the domain (including the first dot) and <number> is the server's transport identifier, as specified in the DOMAIN attribute in the server's configuration file. The client program must print logging information on standard output.

You should appropriately structure your software to clearly separate communication operations from managing the naming entries, such that the communication mechanisms could be easily replaced by others (such as another transport protocol or RPC). Further, the client and server part of the communication mechanism should be separated to facilitate code reuse between the client and the server program. The name server does not have to be implemented as a concurrent server. It is acceptable to process all incoming requests sequentially.

Transport Addressing - Basic

DNS servers operate on standard ports; this works great on the internet but for our the purposes of this assignment would result in a great deal of port collisions while you debug your assignments (and we mark them). To address this, you must devise a way to map from the <transport id> field in the configuration files to the actual port numbers used by the system. The mechanism you use is up to you, although to achieve the 60% level no port number can be hard coded (in the code or a configuration file); to achieve the 100% level the advanced transport addressing scheme (below) must be used. Some possible forms of basic port mapping will be discussed in class.

Transport Addressing - Advanced

In reality, at most one name server would execute on a network node and it would have a fixed transport port where it can be contacted. For the assignment, it must be possible to run multiple name server processes and assign fixed transport identifiers to each of them, so that they can be referred to in the configuration file of other servers. Using fixed port numbers may result in a conflict where multiple users on the same machine attempt to use identical port numbers. Because it would be a significant challenge to rely on the availability of certain port numbers during development and testing, the name servers use an additional level of indirection, borrowed from the RPC system that typically exists on every UNIX machine.

Server processes bind their socket(s) to arbitrary port numbers assigned by the operating system and register the resulting port numbers for their transport identifier with a port mapping service (typically called `rpcbind` process). This mapping service uses two parts, called program number and version number in the RPC system, to identify an actual transport port number. By using a unique identifier as program number (e.g. each student's student ID), the version number can then be used to identify different servers without causing any naming conflict, even if two users use the same numbers to identify server processes on a single machine. Effectively, the used program number (student ID) is globally unique for all servers and clients and the version number is used as transport identifier to distinguish different servers.

A process registers and deregisters a transport port with the port mapping service by using the system functions `pmap_set` and `pmap_unset`. A port is looked up by calling `pmap_getport`. Please see the respective man-pages for details. Example code for C/C++, Java, and Python will be provided on the course web page. Note that on Solaris, port mapping only works, if the process registering the mapping actually has a socket bound to that address (as in the example code). Otherwise, the system automatically removes the mapping later.

IMPORTANT: As a convention, the root server is always assigned transport identifier 1. Therefore, the client program implicitly know how to find the root server to begin the name resolution.

Configuration Example

Example for a root configuration file:

```
DOMAIN      .     1
NS          xyz   15    I
NS          abc   19    R
ID          hello 9999
```

Example for a domain configuration file for the domain .xyz:

```
DOMAIN      .xyz  15
NS          ccc   20    R
ID          fred  7777
ID          mary  8888
```

Given these configuration files, a lookup for `.xyz.mary` starting at the root server would be resolved through one iterative lookup step. It would result in the number `8888`.

Additional Comments/Hints

Note that a client or name server does not request a specific resolution method from another name server. Instead, each server decides locally, based on the `I` or `R` flag for the corresponding edge label, whether it carries out a recursive name lookup or just returns the next server's address.

There is a residual chance that another process on any machine has already registered a port in the port mapping service with the same pair of program number and version number. You can inspect all registered port mappings on a machine by running `rpcinfo -p` in your shell.

Download

Example code for C/C++, Java, Python.

Evaluation

The assignment is to be done individually or in pairs. Your program will be evaluated in the `student.cs` undergraduate computing environment. Your program should not silently crash under any circumstances.

If the code cannot be run for the step you have completed, we cannot assign grades. Be absolutely sure you have validated your code on the `student.cs` environment and have named your scripts / programs as required. This is essential!

Grade Levels

As with the last assignment, the grading has been split into a series of levels. Each level requires that all of the functionality form previous levels be working unless it has been specifically superseded.

Level 1 (20%) - Respond to requests using a single server. Any port configuration (including hard-coded ports) may be used. The server must be able to read the configuration file. The server must be able to respond to multiple (non-concurrent) requests.

Level 2 (40%) - Two server instances must be supported. Only iterative queries will be issued.

Level 3 (60%) - No fixed ports are allowed (although any other means is acceptable); see *Transport Addressing - Basic*. Any number of servers may be used although only iterative queries will be requested. Servers and clients must handle error cases intelligently and writ descriptive messages to the screen to indicate status.

Level 4 (80%) - Full recursive support required (in addition to full iterative support). Logging messages must be written to the screen ad to log files (according to *System Structure*).

Level 5 (100%) - The advanced transport addressing scheme (*Transport Addressing - Advanced*) must be employed. Assignment must be fully functional.

Submitting your assignment

This assignment can be completed individually or in a team of at most two people. Only one copy of the assignment needs to be submitted, but make sure that **both** team members names are on any submitted files. Your source code files / scripts / `README` should be be submitted as a zip file called `<first-last>[_first-last]_a3.zip` (if you are performing the assignment alone obviously the second first-last is not required). Do not submit any object or executable files. Email the assignment by 0830 on Mar 31 to `rth.cs436@gmail.com`. Late assignments will not be accepted.