Remote Machine Information Service

Program Five

CS 3411 Spring 2022

Due: Friday, April 15 at 11pm

Motivation

It can be helpful to learn information about a machine on the network, for example to identify the OS it runs and its architecture, which in turn determines whether a binary can run on that machine. In this project, you will implement a service that allows the user to determine the machine name, operating system and version, and hardware architecture.

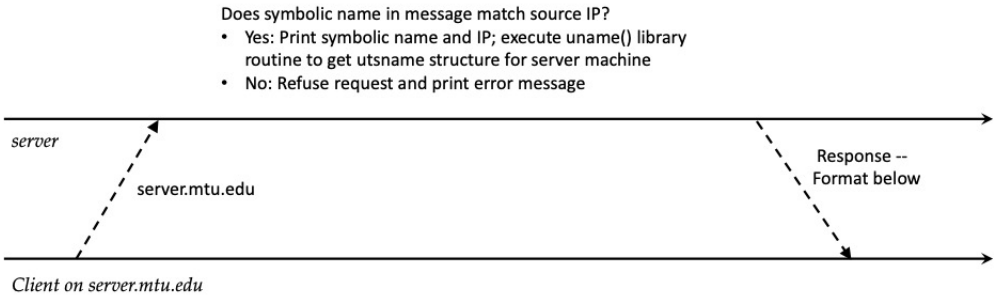
Requirements

The system is comprised of a client named `rinfo` and a server named `rinfod`. You will write both parts.

The client and server communicate using the socket interface and the unreliable UDP protocol. Reliability is achieved through the client, which resends a request until a response is received or the client decides that the server is unavailable. The behavior of the client and server is described in more detail below.

Server

The exchange between client and server is depicted in Figure 1. The server listens for UDP requests on an unchanging port. When a well-formed request is received, the server responds. A well-formed request contains a null-terminated ASCII string that represents the symbolic name of the client. The server resolves the symbolic hostname to an IP address and compares the IP associated with the hostname in the message to the IP address returned by the operating system along with the request message. The server prints out the hostname from the message and the IP when they match and then services the request. The server drops the request and prints out an error message when they do not match.



Server Response Message Format

Field	Bytes	Content
1	4	Binary value (in the network byte order) for the number of characters in the remaining fields
2	Variable	sysname field of the ustname structure
3	Variable	nodename field of the utsname structure
4	Variable	release field of the utsname structure
5	Variable	machine field of the utsname structure

Figure 1: Communication Overview

The server response message contains an unsigned 4-byte integer in the network byte order, followed by four null terminated character strings. The integer value is the number of bytes total in the four strings. (This includes the zero byte that terminates each of the four strings.). There should be no intervening characters between the null-byte that terminates a string and the first character of the following string.

The server determines the string values for the response by running `uname`. Each string represents a field of a `utsname` structure as described by the manual page for `uname` in section 3P of the UNIX manual on departmental Linux systems (e.g. `guardian.it.mtu.edu`). The first string is the name of the implementation of the operating system, the second is the name of the node on the network, the third string is the implementation release level and the last string is the hardware type. See `man 3p uname` for more information. Note that you may have to track down the required header file to identify the fields in a `utsname` structure.

Normally, a service is associated with a particular port. All that is needed to communicate with the server in this case is the hostname. However, since you cannot spawn privileged processes, **you cannot assume any particular port will be available for your server**. Since you don't have root access, I will allow you a single NFS-shared file in which you may write things which allows clients the illusion of a well-known address for the server. The file should be named `serverlocation` and can be assumed to be in the CWD of any process that accesses the file. The file must contain the server's UDP port (as a `short` in the network byte order). The grader may write a program that looks in the CWD for a file named `serverlocation` and reads and outputs the host port, assuming the file contents are in the format given here. **The server should delete the location file and terminate whenever it receives SIGINT or SIGTERM.**

Client

The client is invoked as: `rinfo symbolic-hostname`. It sends a request to a server at `symbolic-hostname`. The client reads the server's port number from a file named `serverlocation` in the CWD. If the file does not exist, the client waits 4 seconds, tries to read the file one more time and then exits if the file cannot be read. The `sleep` and `alarm` calls both allow a process to arrange an action after a specified number of seconds. (See the manual pages.)

The client sends the request to the server and waits 2 seconds for a reply. The request contains the client's symbolic name followed by a zero byte. If a reply is not received in 2 seconds, the client tries one more time and then exits. If the reply is received, each string followed by a newline is written out to stdout. (Write the strings in the order given above). Then the client exits.

Collaboration

Empty hands discussions are allowed for this project.

Testing

Notice that the client and server communicate using a well-defined protocol. Your client should interoperate with the server available at: `jmayo/public/cs3411/projects/p5/server`. This server implements the server protocol (except for signal handling).

Your server should interoperate with the client available at: `jmayo/public/cs3411/projects/p5/client`. This client implements the client protocol.

Submission

Submit a makefile that supports the following directives.

`all`	Creates `rinfod`, `rinfo` and any object code
`clean`	Removes all the created object code and binaries.

Submit a tar file named `p5.tgz`. The file should contain your source code and makefiles. Include a single README that gives your name, the number of slip days that you used for the project and that describes each file in your project directory. Typing the commands: `gtar zxf p5.tgz; make all;` should produce `rinfo` and `rinfod`.