Multi-threaded
DNS Resolver

Guide and Documentation

By Jacob Werner

Contents

Summary

This project is a multi-threaded application that resolves domain names to IP addresses. The application takes a set of name files as input, which contain one hostname per line. Each input file will be serviced by an individual thread and there can be a variable difference between input files and threads. The application will synchronize access to shared resources and there is built-in protection for resources that are not thread-safe by default.

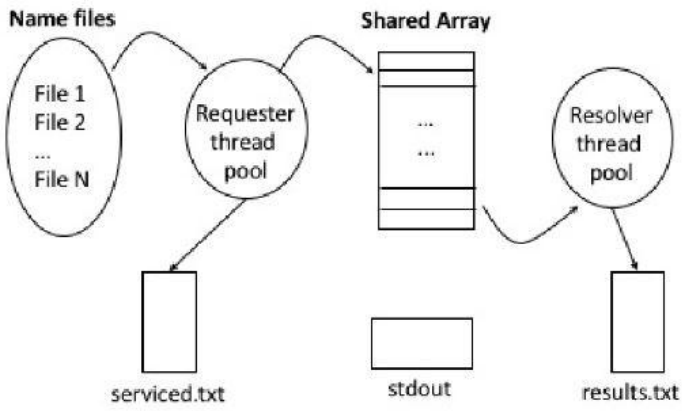
Design Summary

This application is composed of two subsystems, each with a single thread pool: requesters and resolvers. These subsystems communicate through a shared array.

Requester Threads: Requester threads will service a defined set of input text files, each containing hostnames. A requester thread will claim an input file, read hostnames line-by-line, and place each hostname as its own entry into the shared array. Once all input files are serviced, each thread will write the number of input files it serviced into a new file named *serviced.txt.*

Resolver Threads: Resolver threads will service the shared array that is populated by the requester threads. A resolver thread will claim an entry from the array, and call upon the DNS lookup code to query for an IP address. If a hostname can be resolved, the resolver thread will output the hostname and the resolved IP address to a new line in the *results.txt* file.

The DNS lookup utility will be a function that accepts a hostname as an input string and will generate the respective IPv4 address as an output string. Due to the scope of this project, the complexity of a true DNS lookup may be simplified and might not represent current best-practices for DNS resolution.



The application features synchronized access to shared resources to avoid deadlock and busy wait. Mutexes and condition variables are employed to ensure that input files, the shared array, and the output files are properly accessed and that race conditions are prevented. Because of the built in protection, resources that are not thread-safe by default will be compatible with this application.

The application seeks to limit the number of outside resources that it relies upon. The C standard library and standard Linux pthread, number generator, and file i/o libraries are among the only outside resources required for execution.

Note: This application was designed to run on Linux-based systems.

Limitations

The following have been observed or designed to be limitations when executing this application:

- Even if the DNS lookup resolves multiple IP addresses for a host, only 1 will be identified and included in the output file.
- The maximum number of input files that are currently allowed is up to 10. This value can be changed, but behavior is not guaranteed.
- The maximum number of resolver threads is 10. This value can be changed, but ensure that enough threads are available.
- The maximum number of requester threads is 5. This value can be changed, but ensure that enough threads are available.
- The maximum name length for a hostname is 1025 characters, including the new line terminator.
- The maximum IP length is *IPNET6_ADDRSTRLEN*.

Application Layout

The following files are included within this application:

- *multi-lookup.c*
    - Contains all source code relevant to threads, shared arrays, and file input/output
- *multi-lookup.h*
    - Header file for *multi-lookup.c*.
- *util.c*
    - Contains source code for DNS lookup function
- *util.h*
    - Header file for *util.c*.
- *Makefile*
    - Used to compile application code for execution.

  Note:

Installation and Setup

Installation/Build - To install, clone the repository from GitHub onto a Linux-based system. With all files in a directory together, simply use the *make* command with no additional flags or parameters.

Input File Setup - In order to properly run this DNS resolver, proper input file structure must be adhered to. The file name can be customized to the user's liking as long as the file is in *.txt* format. For each website to be resolved, there must be only one hostname per line, and must be in the format of *name.com* (i.e. facebook.com, google.com) and must exclude the www and https prefix.

> **Note:** There is no limit to how many hostnames are in a single file; however, to take advantage of the design of this application and increase performance, it is recommended to evenly spread out hostnames into as many files as allowed or necessary.
>
> **Note:** Input file samples are provided in the *input* directory.

Running the Application

To run the application, run the following command:
`./multi-lookup <#requester> <#resolver> <serviceFile> <outputFile> <inputFolder>...`

- Where `<#requester>` and `<#resolver>` are integers for the number of respective threads requested by the user for the operation.
- Where `<serviceFile>` is a *.txt* file that the application will print files and hostnames it has serviced before DNS resolution.
- Where `<outputFile>` is a *.txt* file that the application will output resolved hostnames and their IP addresses.
- Where `<inputFolder>` is a *.txt* file with the input hostnames, you may add more input files following this parameter.

Troubleshooting

When running the application, errors may occur. Below is a list of checks and errors built into the application to ensure proper and safe usage:

Error	Explanation	Solution
“Input file cannot open”	This error typically results from an invalid file format or a mistakenly inputted parameter	First, ensure that the input files are all in .txt format. If all files are in the correct format, then ensure that all filenames are correctly spelled in the execution command when running the program. If error still occurs, ensure that all syntax within the file is correct.
“Could not resolve *hostname”*	This error occurs when a requested hostname is incorrectly formatted or spelled, or if the DNS lookup could not resolve the hostname.	First, ensure that the website being requested is spelled correctly. If the website is spelled correctly, determine if it is still published or remove the website from the input.

“Too many arguments”	This error occurs when there are too many parameters entered into the execution command when running the application.	First, ensure that the run command being used adheres to the structure and syntax provided in the *Running the Application* section of this guide. Then ensure that there are not too many input files being used. The default maximum is 10. This value can be observed as MAX_INPUT_FILES in *multi-lookup.h*.
“Can only have up to # resolve threads”	This error occurs when the requested number of resolver threads exceeds the maximum resolver threads allowed.	First, ensure that the parameter entered is below the max threads value found in *multi-lookup.h*. The default value is 10 threads. You may change this number if required.
“Can only have up to # request threads”	This error occurs when the requested number of requester threads exceeds the maximum requester threads allowed.	First, ensure that the parameter entered is below the max threads value found in *multi-lookup.h*. The default value is 5 threads. You may change this number if required.
“Invalid service file path” OR “Invalid output file path”	This error occurs when the incorrect path or name for the output/service files is inputted into the run command.	Check for correct spelling and validity of the path for whichever file is throwing the error.
“Failed to open performanceOutput.txt”	This error occurs when there is not a valid file to dump performance data into.	Make sure to create a *.txt* file named *performanceOutput.txt*
“Input directory invalid”	This error occurs when the directory to store the input files cannot be created correctly.	Ensure that the proper input files are included in the run command and that the maximum limit for input files is not exceeded.