S-38.3610 Network Programming Assignment, Phase 2: HttpDns Server

Antti Nilakari 66648T <antti.nilakari@aalto.fi>

Overview

The HttpDns server functionality is implemented by the program httpdnsd. Unlike the client, the server has been completely written in C99. The server supports both foreground and daemonized operation. Serving multiple users simultaneously is achieved through the use of pthreads.

Currently only HTTP/1.1 GET functionality is supported. PUT functionality will be added later on.

Module structure

The program is comprised of following C modules, each comprising of a .h header file and a corresponding .c file:

- Httpdnsd – Main entry point to the program. Handles parsing command line parameters and daemonization.
- Httpserver – Main server logic. Contains the server main loop, and the individual connection handler thread logic.
- Http – HTTP client logic. The server needs HTTP client functionality to register to the *http://nwprog1.netlab.hut.fi:3000/servers-all.txt* list.
- Url – Functionality for parsing URLs. Used by the HTTP client logic.
- String – A custom higher-level string library to allow easier and more efficient string operations, such as appending to a string buffer. Used by httpserver.
- Thread – Helper functionality to spawn pthreads with less code.
- Socket – Wrappers for UNIX socket and file descriptors to handle trivial error cases and facilitate easier usage of TCP sockets. Used by all networking and file handling code.

In addition, the following header-only helper library is used:

- util.h – Contains the VERBOSE macro to facilitate console logging.

No external libraries are used. All program code has been written by the author between 2011 and 2013.

Key functions and operation logic

The operation logic of network functionality can be roughly split to two different parts: the main server listener and individual worker threads.

Pthreads were chosen as the solution to achieving concurrency. The alternative would have been writing a select()-based event handler. Threads were chosen so as to isolate the logic between different client threads and the main listener thread. The threads are run in detached mode and do not communicate back to the main listener thread nor require collecting memory after they have ended.

The main server loop logic is as follows, in pseudocode:

```
Set up signal handlers to handle SIGTERM and SIGINT.
Open a listening socket on given client port.
Register to the nwprog1 server, retrying three times.
While server is running:
     Wait for incoming connection or signal:
     If a connection is incoming:
          Spawn a detached worker thread:
     Else if a signal was caught:
          Server is no longer running.
Deregister from the nwprog1 server, retrying three times.
Close the listening socket.
```

The accept handler requires very little logic, apart from handling incoming signals.

The individual connection handler thread logic is as follows:

```
Read at maximum 8192 bytes
Parse the header to request, header, payload start.
If header fit into the buffer:
     If request is GET:
          Open the file specified in the request
          If file open:
               Send a “200 OK” header
               While content bytes remaining:
                    Read at maximum 8192 bytes from file
                    Write bytes to the client
          Else:
               Send a “404 Not Found” reply
     Else:
          Send a “405 Method Not Allowed” reply
```

```
Else:
     Send a “400 Bad Request” reply.
```

If either the file or network operations in GET handler fails, the connection is closed in the middle of the transaction. There is no mechanism to report mid-I/O transaction errors in the HTTP protocol as far as I know.

Build and installation instructions and requirements:

Requirements:

:

- A C99-compliant compiler such as Clang or GCC
- Standard C library supporting at least POSIX version 2008.09 and Pthreads.

Building:

In the source directory, run:

```
$ make clean all
```

The program should compile without any warnings.

User instructions

The www root directory used for storing and retrieving files is the current working directory when the server is launched.

```
Usage: ./httpdnsd [-f] [-v] PORT
    -f    Stay on foreground
    -v    Print verbose output
    PORT  Port or service name to listen on
```

To enable verbose reporting to stderr, use the -v command line flag.
To prevent daemonization, use the -f flag.
The port may be either in numeric form (e.g. 8080) or a service name (e.g. www). If binding to a low port (< 1024) is desired, run the server with root privileges.

To shut down the server, send a normal SIGTERM or SIGINT signal with KILL. If the server is running on the foreground, hitting ^C is enough.

Testing and known limitations

Testing

The server has been tested using the curl command line utility for downloading and uploading files. The GET method has also been tested with the prior httpdns client and Google Chrome. Runtime testing all the possible failure conditions is almost impossible and thus has not been done; however, to complete any operation all calls are required to complete successfully. No undefined behavior is permitted.

The validity of memory accesses has been verified with Valgrind.

All return values of system calls, including those that do network I/O, are verified. If a call fails for some reason, the server tries to fail gracefully. Http client operations are retried at maximum three times. Failing disk I/O operations result in closing the socket connection.

Known bugs and limitations

Only the GET method is supported at the moment.

Answers to specific questions

1. Concurrency is achieved by running every client request in a different thread. A hanging client will not prevent other clients from receiving CPU time. A hanging client will time out at some point, shutting down the corresponding worker thread. Currently, no file locking is done so a hanging client cannot prevent other clients from doing I/O on the same resource. A so called reader-writer-concurrency pattern will be implemented in the final version.

2. The socket module uses AF_INET6 exclusively. For IPv4 backwards compatibility, AI_ALL and AI_V4MAPPED are used.

3. Not tested

4. Not tested

5. Not tested. However, when registering and unregistering the server I noticed that the remote http server hangs if it receives a PUT request with Content-Length: 0. If an empty file is sent without a Content-length field, the server responds fine.

Diary