LuaFCGId README

October 7th, 2011

Contents

1 Summary

luafcgid is a multithreaded FastCGI server that runs under BSD/Linux. It manages a number of independent, persistent Lua states, that are then loaded with Lua scripts from the file system. These scripts are loaded/initialized on demand, and held in memory for as long as possible. The Lua scripts are also allowed to interface with the FastCGI libraries: thus providing an extremely fast, streamlined and lightweight platform from which to develop web-centric apps in Lua.

2 Support

There is a support forum available at http://forum.luahub.com/index.php?board=17.0

3 License

See the LICENSE file included with the source code

4 Testing

All development testing is done with the following server platform:

4.1 Hardware:

Pentium 4 3.2 Ghz Intel MB chipset Intel Pro+ 1GB ethernet 256MB RAM

4.2 Software:

- FreeBSD - last stable release
- nginx web server - last stable release
- Lua - last stable release
- libthr - drop-in replacement for libpthread

NOTE: Further testing may be done on other platforms when available. Volunteers welcome.

5 Prerequisites

- Lua 5.1
- libfcgi 2.4
- libpthread/libthr/pthreadw32

6 Installation

See the INSTALL file for your platform. If you do not see a INSTALL file for your platform, then you can try to use the generic build by running:

```
# sh configure
# make
```

You may have to modify the Makefile created by the configure script, and will have to install manually. Any platform or distribution-specific Makefiles and initd/rc.d scripts are always welcome and can be submitted by github pull request, or to the support forum:

http://forum.luahub.com/index.php?board=17.0

6.1 Webserver (nginx):

Add the following lines to your server{} section:

```
location ~ \.lua$ {
    # for BSD/Linux
    fastcgi_pass   unix:/var/tmp/luafcgid.sock;
    fastcgi_param  SCRIPT_FILENAME  $document_root$fastcgi_script_name;
    include        fastcgi_params;
}
```

NOTE: *make sure your root directive is set correctly*

7 Design

luafcgid spawns and manages a number of worker threads that each contain an isolated blocking accept loop. The FastCGI libraries provide a connect queue for each worker thread so that transient load spikes can be handled with a minimum of fuss.

```
                  +---------------+
              +-->| worker thread |--+            +-------------+
+----------+  |   +---------------+  |            |   scripts   |
| luafcgid |--+                      +-- mutex -->| loaded into |
+----------+  |   +---------------+  |     |      |  Lua states |
     |        +-->| worker thread |--+     |      +-------------+
     |            +---------------+        |
     |                                     |
     +------------- housekeeping ----------+
```

Lua is then introduced into the picture by created a shared Lua state for each Lua script that is requested. A script can also be loaded into more then one state if heavily requested. All scripts - including duplicates (clones) - are completely isolated from each other. After a state is initialized and loaded with a script, it is kept in memory for as long as possible. This allows for persistence across HTTP requests. Each Lua VM is run within a worker thread as needed. The use of on-demand clones allows for multiple workers to run the same popular script. There is a configurable limit to the total number of Lua states that luafcgid will maintain. When this limit is reached, popularity and aging are used to decide which states to flush and reload with a new script.

Global housekeeping is run on a regular cycle (heartbeat) independent of requests. Current tasks include flushing any loaded scripts that have been modified. More tasks may be added in the future.