MrHTTPD README

Dr. Martin Rogge

2011-12-06

Table of Contents

Welcome

Welcome to MrHTTPD. You can read it as Mr HTTPD if you like, although in reality the name refers to my initials. ;-)

MrHTTPD is maintained and copyrighted by Martin Rogge. The software is licensed to you under the GNU GENERAL PUBLIC LICENSE. The complete wording of the GPL is available in the file COPYING. I would also appreciate if you sent me a short email in case you find this software useful and deploy or modify it for any purpose. My email address is <martin_rogge@users.sourceforge.net>.

The MrHTTPD home page is http://sourceforge.net/projects/mrhttpd.

Before we get into more detail, here is the quick installation guide for the impatient reader:

Quick Installation Guide

1. Review mrhttpd.conf
2. ./configure
3. make
4. as root: make install

NB: you can skip step 2 and 3 if you feel lucky

Credits

MrHTTPD started as a modification of an existing open source project, called GazTek HTTP Daemon (ghttpd for short). The GazTek HTTP Daemon is copyrighted under the GPL by Gareth Owen (<gaz@athene.co.uk>), last available under http://members.xoom.com/gaztek.

In the meantime MrHTTPD has evolved significantly and the source code bears little or no resemblance to ghttpd. Also the runtime behaviour shows fast performance and enterprise strength stability.

The documentation you are reading right now has mostly been written in plain text and incrementally extended over various MrHTTPD releases. Beginning with release 2.1.0 the original documentation has been migrated to DocBook format. This move was prompted by practical as well as professional considerations, since I work in software development and I am currently concerned with questions of text and documentation management. For editing the DocBook version of this documentation I have been using the XML Mind [http://www.xmlmind.com/] software under the personal edition license. The software is highly recommendable for users new to DocBook or XML in general.

I am currently still experimenting with different tool chains to produce suitable output formats from the DocBook original. The output quality is not always to my liking, and I still need to fix the plain text version manually for better readability. Nonetheless I am confident I will solve those issues as I get more familiar with the whole concept. ;-)

Concept & Restrictions

MrHTTPD is a threaded web server that is lightning fast, simple, robust, secure and has a very small memory footprint. The binary is 15 to 20 kilobytes in size, depending on configuration and CPU architecture. MrHTTPD serves files at 3 to 4 times the throughput of Apache and runs CGI scripts. It is not designed to implement the full HTTP protocol. For instance, MrHTTPD does not implement HTTP methods other than GET. Since version 2.0 MrHTTPD supports HTTP Keep-Alive.

Having started as a Linux specific project taking advantage of a few Linux specific featurs, the coding should be portable to other operating systems by now. The only caveat at this stage is: it has never been tested by me. ;-)

Background & History

The reason I started working on MrHTTPD was that I needed a web server providing simple services in my home local area network. In particular, it had to serve static files and run Perl scripts on my internet router which was a self-made and highly specialised Linux system. In fact, I had previously installed the Apache web server on the internet router and run it for years. Functionally, I have never had any reason to complain about Apache. The only reason for moving away from Apache is that it is an over-dimensioned solution for my requirements.

Not knowing much about socket programming in the beginning, I started experimenting with various open source web servers. Whilst many of them were inherently unstable and of bad performance I could learn about the basic structure of a web server. This excercise led to the first usable version of MrHTTPD which was rock solid and performing well in my local area network.

Eventually competitiveness got the better of me and I compared benchmark results with Apache. The astounding result was, despite being so much more concise, MrHTTPD would only reach half the speed of Apache in serving files. The reason was, MrHTTPD would fork a new process for every incoming connection, that is one for every GET request. Apache, on the other hand, relies on a number of already forked worker processes and distribution of work via inter-process communication. So I took up the gauntlet, did some reading about threads and finally transformed the server engine into a fully threaded design. The result was a speed increase by more than factor four when serving static files. MrHTTPD was able to outperform the Apache server even when the latter used HTTP Keep-Alive (a feature that MrHTTP offers only since version 2.0).

Note, however, that the high speed transfer only applies to serving static files. CGI programs are always started in a separate process (both by Apache and MrHTTPD) and they are usually time consuming. I have not implemented other protocols like FastCGI because so far I had no use for it.

Installation

MrHTTPD is configured at compile time and only at compile time. This (and the small size) makes it suitable for embedded systems.

In order to configure MrHTTPD you load the file mrhttpd.conf into an editor. The file distributed with MrHTTPD contains explanations of the possible settings as part of the commentary.

In general, the configuration file defines pairs of variables and their values separated by a "=" character. The configuration file is included in bash scripts and make files. This means the file must fulfil the syntactical requirements of both applications. For instance, bash requires that there be no spaces around the "=" sign.

Let us now go through the list of variables. Note that some of these variables have become optional in v2.0 and will either default to some useful value, or disable the compilation of the associated feature. For details please refer to the sample configuration file.

SYSTEM_USER — defines the user account under which MrHTTPD will run. Note: If MrHTTPD is configured to listen at a non-privileged port it need not be started as root. However, if it is started as root you should specify a user with restricted access rights here.

SERVER_ROOT — specifies the directory that will become the chroot jail of the server. All other paths with the exception of BIN_DIR are therefore relative to SERVER_ROOT. Be aware that a chroot jail can be very restrictive. In particular all your document files and all binaries and libraries required for running external programs must be replicated in the chroot jail.

SERVER_PORT — defines the TCP port the server is listening on. The standard for the HTTP protocol is port 80, but you can use any other port if you like. It is quite possible to test one version of MrHTTPD on a test port while the productive web server listens on port 80 at the same time.

SERVER_NAME — defines the name of the server in the sense of the host name. This is different from the "Server" attribute in the HTTP header which is hardcoded to "MrHTTPD/2.2.0".

SERVER_DOCS — defines the root directory for internal files. Internal files are documents like the 404 error page.

BIN_DIR — defines the installation directory for the binary. This is the only directory not affected by SERVER_ROOT since it is not runtime relevant.

DOC_DIR — defines the root directory for all static files (HTML, CSS, JPG, etc.), ie. the actual productive content of the web server.

CGI_URL — defines the URL component indicating that the resource is a CGI script rather than a static file. Whenever MrHTTPD finds this component at the beginning of the resource path, it assumes the resource is an executable CGI script.

CGI_DIR — defines the root directory for CGI scripts.

DEFAULT_INDEX — defines the name of the default file in a directory. Typically you use "index.html" or similar. MrHTTPD will serve this file if no file name is specified in the URL. For example, if a client requests http://server/path/ MrHTTPD will try to send http://server/path/index.html.

PRAGMA — specifies an optional Pragma parameter that is sent with every HTTP reply. A typical value is "no-cache" if you want to suppress caching by proxy servers and frontends.

LOG_LEVEL — determines the amount of information saved in the log file. Note: the higher the level the more code is generated. If you specify a log level 0, all routines related to log generation, file handling etc. are omitted from the compilation process.

The meaning of the values is as follows:

LOG_LEVEL=0 no log entries are created.

LOG_LEVEL=1 additionally, all rejected requests are logged

LOG_LEVEL=2 additionally, all accepted requests are logged

LOG_LEVEL=3 additionally, every unsuccessful reply is logged

LOG_LEVEL=4 additionally, every successful reply is logged

LOG_FILE — defines the file name used for saving the logs.

EXT_FILE_CMD — specifies a binary that is used to determine the mime type of a resource. A typical binary would be the file(1) command. Note that the external file command is called only after evaluating the file suffix according to a list compiled into MrHTTPD. The obvious reason is the performance impact of an external call.

The external file command is always called with parameters "-b --mime-type". If you do not want that, you need to specify a shell script stripping the first two parameters.

Note: the external file command and all its dependencies are relative to SERVER_ROOT. If you use SERVER_ROOT you will probably need to copy the following files into the chroot jail (your mileage may vary):

usr/bin/file

usr/lib/libmagic.so.1

usr/lib/libz.so.1

lib/ld-linux.so.2

lib/libc.so.6

etc/file/magic/*

SENDFILE — chooses which implementation is used for sending files. The choice is between a routine in user space and the kernel function sendfile(). Whilst the latter promises better performance and lower cpu load, you might want to choose the user space routine for trouble shooting. Also, on operating systems other than Linux you need to select the user space routine.

QUERY_HACK — is an option for the method of processing query strings. If this variable exists the query string of a resource will be interpreted as part of the file name, provided the resource begins with the string in QUERY_HACK.

Example: assuming a request for a resource `index.html?sorted=yes`. Normally the server will read the file `index.html`. The query string

`sorted=yes` will be omitted. If the resource is a CGI script the query string will be passed in the environment string QUERY. If you specify QUERY_HACK=index, the processing will be different and the server will read the file `index.html?sorted=yes`.

The option is named a hack because it is a violation of the HTTP specification. It is, however, useful when you have mirrored dynamic resources using `wget`, and `wget` has created filenames containing the query part.

After the configuration step say

```
./configure
```

or

```
make config.h
```

Either command will start the script "configure" which will do some basic sanity checks on mrhttpd.conf and create the C include "config.h".

After that you may call

```
make
```

and finally

```
make install
```

NB: You need to be root for the latter. As an experienced user you may feel safer to look at the Makefile and do the installation manually. It is not rocket science.

Starting and Stopping

MrHTTPD is always started in standalone mode and without parameters. It will send itself into the background. The foreground process will exit immediately. You can now do a test run from a local web browser by pointing it towards http://localhost/ (or whatever server, port and resource is appropriate in your case).

MrHTTPD will not read any configuration file at runtime. All parameters have been compiled into the binary. For that reason MrHTTPD will accept a SIGHUP signal but it will not do anything.

MrHTTPD can be stopped by sending it the SIGINT or the SIGTERM signal, ie. you can say any of the following:

kill -2 <pid>

kill -15 <pid>

killall -2 mrhttpd

killall -15 mrhttpd

After receiving those signals, mrhttpd will release the socket and wait 5 seconds for threads and child processes to finish. So you can start another server listening on the same port straight away. Please avoid sending a SIGKILL signal unless there is a good reason. A typical control script called rc.httpd is provided with mrhttpd. If you want to use it in a standard location like /etc/rc.d/ you need to copy it manually.

```
#!/bin/bash
#
# Start the web server
#

case "$1" in
   'start')
      /usr/local/sbin/mrhttpd
      ;;
   'stop')
      killall -2 mrhttpd
      ;;
   'restart')
      killall -2 mrhttpd
      sleep 1
      /usr/local/sbin/mrhttpd
      ;;
   *)
      echo "usage: $0 start|stop|restart"
esac
```

Performance

Performance testing is a difficult topic. Particularly when you venture into extreme load scenarios.However, in order to detect regressions and ensure stability in normal operations I now routinely execute a high load performance test script as part of every release of MrHTTPD.

For release 2.2 I have run the tests on my main development machine which has an Intel Core i3-530 CPU at 3500 MHz and 4GB of RAM. The CPU offers two cores with hyperthreading, resulting in 4 logical processors. The operating system was Slackware 13.37 (plus a few upgrades from Slackware Current). I have run all tests on two kernels, namely on vanilla 3.1.4 and on 3.1.4-ck2, the latter containing the BFS scheduler and the interactivity patches provided by Con Kolivas [http://ck-hack.blogspot.com/].

If you want to repeat my performance tests, please observe that logging has a noticable effect on the results under the extreme work loads created by the test. The results shown here have been obtained with logging disabled for all tested HTTP servers. I have included the test script `perftest.pl` in the subdirectory `extra` of the distribution. It requires the benchmark program `ab` which is part of the Apache distribution.

I have included the raw performance data obtained from the web servers Apache, Lighttpd and MrHTTPD in CSV format (comma separated values), as produced by the test script `perftest.pl`. I have also included a version in ODS format, as used by LibreOffice or OpenOffice Calc. The ODS version is particularly useful for filtering rows.

As I mentioned in the beginning, performance testing under extreme load is a difficult topic. I found I had to push certain system limits prior to be able to complete the test suite for concurrency levels above 1000 (which for the machine in question is way beyond healthy anyway). In particular I needed to run the following commands beforehand:

```
echo 65536 > /proc/sys/net/ipv4/tcp_max_syn_backlog
echo 65536 > /proc/sys/net/core/somaxconn
echo 65536 > /proc/sys/net/core/netdev_max_backlog
echo 16777216 > /proc/sys/net/core/rmem_max
echo 16777216 > /proc/sys/net/core/wmem_max
echo 4096 87380 16777216 > /proc/sys/net/ipv4/tcp_rmem
```

```
echo 4096 16384 16777216 > /proc/sys/net/ipv4/tcp_wmem
echo 16384 > /proc/sys/vm/min_free_kbytes
ulimit -n 65536
ulimit -i 65536
ulimit -u 65536
```

But even then the kernel interfered when running the performance test for Apache and Lighttpd: the connection was dropped and a system message "Possible SYN flooding" was stored in the system log. From the kernel sources it appears this message is generated when the incoming connections are not accepted quickly enough by the listening server. Be it as it may, I could not find a solution and the corresponding results are therefore missing from the result set.

Looking at the results, it can be said that Apache is trailing behind in virtually all tests. For CGI scripts, Lighttpd seems to be the best performer overall, particularly on the vanilla kernel.

For static HTML pages, particularly when connections are kept alive, MrHTTPD wipes the floor with the competition. As an example look at the following result reading 1000000 static files at concurrency level 100:

Kernel Version	**Server**	**Keep-Alive**	**Requests per second**	**Rate**	**Max. time [ms]**
3.1.4	Apache/2.2.21	yes	25427	33388	265
3.1.4	lighttpd/1.4.29	yes	54741	68892	6
3.1.4	MrHTTPD/2.2.0	yes	88898	100888	5
3.1.4-ck2	Apache/2.2.21	yes	43760	57462	17
3.1.4-ck2	lighttpd/1.4.29	yes	49227	61652	6
3.1.4-ck2	MrHTTPD/2.2.0	yes	163158	185150	2

The throughput offered by MrHTTPD is three to four times higher than Apache and Lighttpd. Result sets at higher concurrency continue this trend, and at insane concurrency levels like 10000 or 20000 MrHTTPD is the only responsive server remaining.

The other stunning fact proven by these figures is the significant impact of the BFS CPU scheduler and other optimizations inherent in the CK patch, leading to almost twice the throughput compared to the vanilla kernel (which was btw configured to use the CFQ scheduler). This is even more surprising as the design goal of the CK patch is desktop interactivity rather than server throughput.

It is also noticable that a fully threaded design like MrHTTPD benefits particularly well from the CK patch, whereas the event-driven design of Lighttpd often seems to perform slightly better under the vanilla kernel.