Last update: 08 Jan 2021

Motsognir – The mighty gopher server

User manual

Written by Mateusz Viste

http://motsognir.sourceforge.net

Table of Contents

Introduction....3
Installation....4
Configuration file....5
Directory listings....10
Gophermaps....11
  Files listing....12
  Default (system-wide) gophermap....12
  Sub-gophermap scripts....12
Per-user gopher directories....14
CGI support....15
  Let's see how CGI works....15
  How to enable CGI support in Motsognir....16
PHP support....17
  How to enable PHP support in Motsognir....17
  Example....17
Caps.txt support....18
Serving different content on multiple domain names....19
Plugin development....20
  Queries filtering....20
  Configuration example....20
Security considerations....21
  Run Motsognir as a non-privileged (non-root) user....21
  Choose your file permissions wisely....21
  Use paranoid mode if you are (really) paranoid....21
  Trap the daemon inside a chroot jail....22
Frequently asked questions (FAQ)....23
Legal mumbo-jumbo....24

Introduction

Motsognir is a robust, reliable and easy to install open-source gopher server for Unix-like (POSIX) systems. Motsognir is a standalone daemon that supports server-side CGI applications and PHP scripts, is plainly compatible with UTF-8 filesystems, and is entirely written in ANSI C without external dependencies. It supports both IPv4 and IPv6 'out of the box', without requiring any special configuration.

Gopher is a distributed document search and retrieval network protocol designed for the Internet. Its goal is to function as an improved form of Anonymous FTP, enhanced with hyperlinking features similar to that of the World Wide Web.

I wrote Motsognir primarily for fun, but it appears to have become a strong, full-featured gopher server implementation with some very nice points: easy to install, lightweight, portable, secure...

The Motsognir gopher server is meant to be used for small projects (like home servers), but should scale well on bigger architectures just as well. All the configuration is done via a single configuration file, which has very reasonable defaults. That makes Motsognir easily maintainable, and allows the administrator to have full knowledge of what features are allowed/enabled on the server.

Installation

Installing Motsognir is very easy on most systems, since it comes packaged as a part of the operating system on many Linux and BSD distributions. However, if your operating system doesn't provide a plug & play motsognir package, then you will have to install it manually.

In the first place, you will have to build it. Assuming you have the basic GNU development tools installed (make, gcc), building Motsognir should be as straight-forward as:

```
# make
```

This will generate a 'motsognir' executable, that you will need to install on your system, then:

```
# make install
```

Now, all you have to do is edit your /etc/motsognir.conf file to make it suit your needs, and then launch motsognir:

```
# /usr/sbin/motsognir
```

Note: If you wish to store the motsognir.conf file outside of /etc/, or under a different filename, then you will have to instruct motsognir about its location every time your run it:

```
# /usr/sbin/motsognir --config /my/custom/location/motsognir.conf
```

Alternatively, you could also modify the default location of the configuration file during compilation, using the -DCONFIGFILE=/etc/somefile.conf compile-time parameter.

Configuration file

Motsognir's configuration file is located by default in /etc/motsognir.conf (unless the default location has been modified at compile-time using the -DCONFIGFILE define), and should be readable by the user which will run the motsognir service. The configuration file is a plain-text file, containing some tokens with values. All lines beginning with the "#" character are ignored (and can be used to put some comments in the configuration file).

If any of the parameter is missing from the configuration file, or have an empty value, Motsognir will use a default value instead. Here below is an example of a self-explanatory configuration file.

```
#####################################################################
#                                                                   #
#     CONFIGURATION FILE FOR THE MOTSOGNIR GOPHER SERVER            #
#                                                                   #
# This configuration file controls how the motsognir gopher         #
# server should behave. Every option listed here can be             #
# overloaded through command-line. Example:                         #
#   $ motsognir --gopherport 7070                                   #
#                                                                   #
# motsognir can run without a config file at all, provided          #
# that it gets its configuration through command-line               #
# options. To disable the config file lookup, use the               #
# command-line --config parameter with an empty argument:           #
#   $ motsognir --config '' --gopherport 7070 etc                   #
#                                                                   #
#####################################################################


## Server's hostname ##
# The hostname the gopher server is reachable at. This setting is highly
# recommended, because the gopher protocol is heavily relying on
# self-pointing links. If not declared, the server's IP address will be used
# instead.
#GopherHostname=gopher.example.com

## Gopher TCP port ##
# The TCP port on which the public Gopher server listens on.
# Usually, gopher servers are published on port 70. Default: 70.
GopherPort=70

## Bind on IP address ##
# Set this 'bind' parameter to the IP address you'd like Motsognir to listen
# on. Note, that this parameter must be either an IPv6 address, or an IPv4
# address written in IPv4-mapped IPv6 notation (for ex. "::FFFF:10.0.0.1").
# If not specified, Motsognir will listen on all available IP addresses.
# This setting might be useful if you have a multihomed server, and you would
# like to serve different content for each IP address.
```

```
# Examples:
#  bind=2001:DB8:135:A0E3::2
#  bind=::FFFF:192.168.0.3
bind=

## Disable IPv6 support ##
# Set this to 1 to DISABLE IPv6 support within Motsognir. Please note that
# when IPv6 is enabled (which is the default), Motsognir may or may not listen
# to both IPv4 and IPv6 sockets. This is somewhat of a mess, and it is related
# to how your operating system treats IPv6 sockets. Most sane operating
# systems support dual-stack sockets. In such environments, Motsognir will
# open an IPv6 socket and mark it as "not-only-ipv6" (setting IPV6_BINDV6ONLY
# to false), thus instructing the operating system to accept both IPv4 and
# IPv6 packets on this socket. Unfortunately, some exotic systems are
# religiously against dual-stack sockets (as of 2019, I know about at least
# two such systems: OpenBSD and DragonFlyBSD). On these systems, an IPv6
# socket is unable to accept IPv4 packets, hence Motsognir ends up receiving
# exclusively IPv6 traffic. As a workaround for such systems, one should run
# two instances of Motsognir (one for each protocol). This means two separate
# configuration files: one with and one without the setting below being set.
disableipv6=0

## Root directory ##
# That's the local path to Gopher resources. Note, that if you use
# a chroot configuration, you must provide here the virtual path
# instead of the real one.
# The default path is /var/gopher/
GopherRoot=/var/gopher/

## Allowed public directories ##
# In specific situations, it may happen that you'd like to be able to serve
# files from outside of your gopher root (typically, if you used symlinks
# inside your gopher root, that points to other places of the file system).
# By default Motsognir won't allow such resources to be served, since requests
# that try to access anything outside of the gopher root are considered as
# potentially malicious. However, if you do want to serve content from outside
# your gopher root, then fill in below the list of directories that are
# allowed to be served. Items of this list should be separated by a ':' char.
# Example: PubDirList=/srv/files:/var/lib/stuff:/tmp
PubDirList=

## User home directories ##
# If you'd like to serve gopher content from user directories, using the
# classic /~user/ URL scheme, then define the user directories location here
# below. The configured location must contain a '%s' tag, which will be
# substituted with the username by motsognir. This must be an absolute path.
# If nothing is configured, then support for serving content from user
# directories won't be available. Example:
# UserDir=/home/%s/public_gopher/
UserDir=

## chroot configuration ##
# If you'd like to put Motsognir inside a chroot jail, configure here
# the chroot directory that shall be used.
# By default no chroot is used.
```

```
chroot=

## Plugin ##
# Power-admins might want to craft some additional logic into Motsognir. This
# is possible using a 'plugin', ie. a simple application or php script that
# Motsognir will submit incoming queries to. The plugin can decide whether or
# not it wants to handle a given query (if not, then Motsognir will process it
# as usual. The queries that Motsognir will submit to the plugin can be
# filtered by using a 'PluginFilter'. This is a 'POSIX extended' regular
# expression that will be compared to every incoming query, and only matching
# queries are submitted to the plugin. Read more in the manual.
Plugin=
PluginFilter=

## Paranoid mode ##
# For paranoidal security, you might want to enable "Paranoid mode". In this
# mode, Motsognir accepts to serve only files with permissions set to "world
# readable".
# Possible values: 0 (disabled) or 1 (enabled). Disabled by default.
ParanoidMode=0

## Activate the verbose mode ##
# Here you can enable/disable the verbose mode. In verbose mode,
# Motsognir will generate much more logs. This is useful only in
# debug situations.
# Possible values: 0 (disabled) or 1 (enabled). Disabled by default.
Verbose=0

## Syslog facility ##
# Motsognir logs all its messages through the LOG_DAEMON syslog facility by
# default. In some situations you may want to change the logging facility to
# a custom one. Setting LogFacility to an integer between 0 and 7 will make
# motsognir log its messages through the syslog facility LOCAL0-LOCAL7.
LogFacility=

## CGI support ##
# The line below enables/disables CGI support. Read the manual
# for details.
# Possible values: 0 (disabled) or 1 (enabled). Disabled by default.
GopherCgiSupport=0

## PHP support ##
# There you can enable PHP support.
# Possible values: 0 (disabled) or 1 (enabled). Disabled by default.
GopherPhpSupport=0

## Sub-gophermap scripts ##
# If you'd like to use sub-gophermap scripts in your gophermaps, set
# SubGophermaps.
# Possible values: 0 (disabled) or 1 (enabled). Disabled by default.
SubGophermaps=0

## Period-stuffing and period-terminator for text files ##
# RFC 1436 mandates that text files returned by a gopher server must feature
# a dot terminator (a single period on a line on its own), and that any dot
```

```
# appearing at the start of a line shall be doubled.
# Many (most?) gopher clients do not follow these requirements, and fail to
# process such extra periods. Enabling 'NoTxtPeriod' will prevent motsognir
# from adding any such periods. Please note that while this may have a
# practical value, it is a blatant violation of RFC 1436.
NoTxtPeriod=0

## Secondary URL-delimiting char
# By default, only the '?' char is recognized as a delimiter between an
# object and the query that must be run on the object. With this parameter,
# you can define an additional character that will be equivalent to '?'. This
# character must be provided in a numerical form, as an ASCII value.
# Example for the hash (#) character:
# SecUrlDelim=35
SecUrlDelim=

## Run as another user ##
# If you start motsognir under a root account, you might want to configure
# it so it drops root privileges as soon as it doesn't need them anymore and
# switches to another user. This is recommended for increased security,
# unless you already run Motsognir as a non-root user.
# To do so, provide here the username of the user that Motsognir should run
# as. Default = no value.
RunAsUser=

## Default gophermap #
# If you wish that your server would use a default gophermap when displaying
# a directory that do not have a gophermap, you can specify here a path to
# the gophermap file you'd like to use.
DefaultGophermap=

## HTTP error file
# When Motsognir receives a HTTP request, it answers with a HTTP error,
# along with a html message indicating why it is wrong. If you'd like to use
# a custom html file, you can set it here. Note, that the specified file is
# loaded when Motsognir's starts. If you modify the file afterwards, you'll
# need to restart the Motsognir process for the file to be reloaded.
# Example: HttpErrFile=/etc/motsognir-httperr.html
HttpErrFile=

## Caps.txt support ##
# Caps.txt is a specific file-like selector, which allows a gopher client to
# know more about the server's implementation (for example what the path's
# delimiter is, where is the server located, etc). When enabled, Motsognir
# will answer with caps-compatible data to requests for "/caps.txt".
# Caps support is enabled by default (CapsSupport=1).
CapsSupport=1

## Caps additionnal informations ##
# If Caps support is enabled, you can specify there some additional
# informations about your server. These informations will be served
# to gopher clients along with the CAPS.TXT data.
# Example:
#  CapsServerArchitecture=Linux/i386
#  CapsServerDescription=This is my server
```

```
#  CapsServerGeolocationString=Dobrogoszcz, Poland
#  CapsServerDefaultEncoding=UTF-8
CapsServerArchitecture=
CapsServerDescription=
CapsServerGeolocationString=
CapsServerDefaultEncoding=

## Extension to filetype mapping ##
# Motsognir looks at file's extensions to advertise the proper gopher resource
# type. If the default mapping is not suiting you, you can load a custom
# mapping using a separate configuration file called an 'extmap', and declare
# it below. The extmap file is a simple text file, where every line provides
# a mapping for a single file extension, in such format:
#  txt:0
#  pdf:P
#  gif:g
# Note: Extensions in the extmap file are processes in a case-insensitive way.
ExtMapFile=

# [End of file here]
```

Directory listings

As any other gopher server, Motsognir will present to gopher clients listings of available directories with a specific presentation. A specific requirement of the Gopher protocol is that it needs to provide a "type" for every resource. To detect that gopher type, Motsognir is simply basing on the file's extension. Below is a table containing the default relations between gopher filetypes and real file extensions, as used by Motsognir:

Gopher type	Description	Files binded to this gopher type
0	Plain text file	*.txt
1	Directory listing	All directories
2	CSO search query	-
3	Error message	-
4	BinHex encoded text file	-
5	Binary (PC-DOS) archive file	-
6	UUEncoded text file	-
7	Search engine query	-
8	Telnet session pointer	-
9	Binary file	All files which doesn't fit into any other category
g	GIF image file	*.gif
h	HTML file	*.htm, *.html
i	Informational message	-
I	Image file (other than GIF)	*.jpg, *.jpeg, *.png, *.bmp, *.pcx, *.ico, *.tif, *.tiff, *.svg, *.eps
s	Audio file	*.mp3, *.mp2, *.wav, *.mid, *.wma, *.flac, *.mpc, *.aiff, *.aac
P	PDF file	*.pdf
M	MIME encoded message	-
;	Video file	-

Note, that the above relations can be overwritten using a custom extension mapping ('extmap') file, declared within Motsognir's configuration file using the ExtMapFile directive. Example:

```
ExtMapFile=/etc/motsognir.extmap
```

Gophermaps

There are situations when you would like to have the absolute control on how the server will display a directory. That's why Motsognir supports gophermaps. If Motsognir finds a file called "gophermap" (without any extension) in a directory, then it doesn't check the directory content, and simply outputs to the user the content of the gophermap. Note, that if you enable CGI and/or PHP support, Motsognir will also look for respectively gophermap.cgi and gophermap.php files. A gophermap file contains gopher entries as described by the RFC 1436. There's an example of a gophermap file (of course <tab> have to be replaced by real tabs):

```
iWelcome to my gopher server!<tab>fake<tab>null<tab>0
i<tab>fake<tab>null<tab>0
0About my server<tab>/about.txt<tab>mygopher.domain.net<tab>70
1Download<tab>/download<tab>mygopher.domain.net<tab>70
1A link to a friend's server<tab><tab>friend.domain.net<tab>70
hMy Website<tab>URL:http://mywebsite.com<tab><tab>
```

You can omit the server's address and server's port parts in your gophermap files. If you don't specify a port, Motsognir provides the one your server is using (usually 70). If you don't specify a host, Motsognir provides your server's hostname. If you specify a relative selector (not beginning by a / character) instead of an absolute path, Motsognir resolves it using the path of the currently browsed directory (but only if the host part is omitted, or pointing to your own server).

Therefore, a simpler form of the above gophermap could look like that:

```
iWelcome to my gopher server!
i
0About my server<tab>about.txt
1Download<tab>download
1A link to a friend's server<tab>friend.domain.net<tab>70
hMy Website<tab>URL:http://mywebsite.com
```

The gophermap file can also contain lines that start with the '#' (hash) character. Such lines are treated as an internal comment, and will not be rendered.

Files listing

A specific feature of Motsognir regarding gophermap files is its ability to generate a dynamic file listing inside a gophermap, using special %FILES% and %DIRS% directives. Example:

```
iWelcome to my gopher server!
i
0About my server<tab>about.txt
i
iBelow are all items I have in this directory:
%FILES%
i
iEnjoy!
```

The %FILES% directive outputs a listing of all files and directories in the current folder, while %DIRS% outputs only sub-directories.

Default (system-wide) gophermap

Motsognir provides you with a feature that allows you to set a server-wide gophermap to be used by any directory that do not have its own gophermap. This is the 'default' gophermap. The default gophermap have to be declared in the Motsognir's configuration file, via the 'DefaultGophermap' directive.

Sub-gophermap scripts

Another feature of Motsognir is its ability to run scripts from within existing gophermaps. Such scripts are called “sub-gophermap scripts”, because they are supposed to output a partial gophermap that will be inserted into our actual gophermap. A sub-gophermap script must be declared in the gophermap with a '=' gopher type. Here below is an example of how such sub-gophermap script would be called:

```
iHello, World! My current uptime is:
=/bin/gopher-uptime.sh
i
```

Note, that for sub-gophermap scripts to run, Motsognir must be configured to allow their execution, via the SubGophermaps configuration directive:

```
SubGophermaps=1
```

It's also worth noting that sub-gophermap scripts will be executed regardless of their extension, as long as their executability bit is set. Hence no need to name them with a *.cgi extension.

Sub-gophermaps also require either CGI or PHP to be enabled (depending on what kind of file the sub-gophermap is).

Per-user gopher directories

On systems with multiple users, each user can be permitted to have public gopher content in their home directory using the UserDir directive. Visitors that consult a URL like gopher://example.com/1/~username/ will get content for the gopher home of the user "username" out of the subdirectory specified by the UserDir directive.

Note that, by default, access to such per-user directories is not enabled. You can enable access when using UserDir by setting a line like this:

```
UserDir=/home/%s/public_gopher/
```

Note, that the UserDir configuration must obey a few rules:

- it must be an absolute path (ie. it has to begin with a '/'),
- it must contain a '%s' tag. This tag is replaced with the username by Motsognir.

CGI support

Motsognir supports CGI applications, that allow to run custom scripts and programs interacting with gopher clients.

Let's see how CGI works.

Each time a client requests the URL corresponding to your CGI program, the server will execute it in real-time, then the output of your program will go more or less directly to the client. In fact, when it comes to answer to the client, the CGI application will output a gopher response (ie. a plain text file for file type #0, a directory listing for file type #1, etc...). This response will be catched by Motsognir, and forwarded to the gopher client as being the request's answer.

The Motsognir gopher server provides some information to the CGI application, by setting some environment variables. Note, that for security reasons – and unlike some other CGI implementations - Motsognir will never feed CGI scripts with any command-line parameters.

Motsognir will set several environment variables, which can be read by the called CGI script. Here is the complete list of these variables:

QUERY_STRING	The URL parameters or query, as provided by the client
QUERY_STRING_URL	The client's URL parameter
QUERY_STRING_SEARCH	The client's search query
SERVER_SOFTWARE	The name and version of the server software
SERVER_NAME	The server's hostname, DNS alias, or IP address, used for self-referencing links
GATEWAY_INTERFACE	The revision of the CGI specification, as supported by the server
REMOTE_ADDR	The IP address of the remote client
REMOTE_HOST	Same as REMOTE_ADDR
SCRIPT_NAME	Script name (for self-referencing links)
SERVER_PORT	The port number to which the request was sent

Note, that the QUERY_STRING variable will contain data inputed by the user. For type #7 items, it will contain the search string (on type #7 items, the gopher client usually asks the user for a query, using some kind of pop-up). For any other item's type, the QUERY_STRING variable will contain the part of the URL after the first "?" character (if any). For example, for a request on "gopher://mygopher.server.com/0/myprog.cgi?hellothere", the QUERY_STRING variable will contain the data "hellothere".

It is also possible to use search items (type #7) with a "?" URL - in such case, the CGI script will be able to read both queries separately via QUERY_STRING_URL and QUERY_STRING_SEARCH.

How to enable CGI support in Motsognir

If you would like to use CGI applications on your Motsognir server, you will have to enable CGI support in the Motsognir's configuration file (GopherCgiSupport = 1). You will also have to make sure that your CGI programs are specifically named with a *.cgi extension (the only exception being CGI scripts called from within gophermaps – these are executed regardless of their actual extension).

PHP support

PHP is a very popular scripting language in the web world. You can use it with gopher, too! Motsognir provides PHP support since its v1.0. The PHP concept is very similar to CGI (historically, PHP was in fact born as a set of custom CGI scripts), therefore you are advised to read the chapter about CGI first. Most of it applies to PHP as well.

The main difference is that instead of trying to directly execute PHP files, Motsognir will feed them to your system's php interpreter, and collect the result.

To pass data to your PHP application, you will have to rely on the QUERY_STRING environment variable.

How to enable PHP support in Motsognir

First of all, make sure that php is available on your system. Then, simply enable the parameter in Motsognir's configuration file (GopherPhpSupport = 1). Note, that all your php files must have the extension *.php to be recognized by Motsognir.

Example

Here below is a simple example of a PHP file that could be used as a dynamic (PHP) gophermap with Motsognir.

```
<?php

echo "iHello, this is a php-driven gophermap\tx\tx\t0\r\n";
echo "i\tx\tx\t0\r\n";
echo "iCurrent date is " . date(DATE_RFC822) . "\tx\tx\t0\r\n";
echo "iServer powered by {$_SERVER['SERVER_SOFTWARE']}\tx\tx\t0\r\n";
echo "i\tx\tx\t0\r\n";
echo "1Go back\t\t{$_SERVER['SERVER_NAME']}\t{$_SERVER['SERVER_PORT']}\r\n";

?>
```

Caps.txt support

Motsognir supports caps.txt since version 0.99.1. Caps.txt is a file-like selector, which allows a gopher client to know more about the server's gopher implementation (like what is the path delimiter character, how are structured server's paths, what the server's location is, etc).

Caps.txt support is configurable via the Motsognir's configuration file, using following tokens:

```
## Caps.txt support ##
# Caps.txt is a specific file-like selector, which allows a gopher client to
# know more about the server's implementation (for example what the path's
# delimiter is, where is the server located, etc). When enabled, Motsognir
# will answer with caps-compatible data to requests for "/caps.txt".
# Caps support is enabled by default (CapsSupport=1).
CapsSupport=1

## Caps additionnal informations ##
# If Caps support is enabled, you can specify there some additional
# informations about your server. These informations will be served
# to gopher clients along with the CAPS.TXT data.
# Example:
#  CapsServerArchitecture=Linux/i386
#  CapsServerDescription=This is my server
#  CapsServerGeolocationString=Dobrogoszcz, Poland
CapsServerArchitecture=
CapsServerDescription=
CapsServerGeolocationString=
```

If you would like to have full access to what Motsognir sends in Caps.txt data, then you might consider disabling the caps.txt support in Motsognir (CapsSupport=0), and simply host your own caps.txt file in the server's root. Here is an example of such custom caps.txt file:

```
CAPS
CapsVersion=1
ExpireCapsAfter=3600
PathDelimiter=/
PathIdentity=.
PathParent=..
PathParentDouble=FALSE
PathKeepPreDelimeter=FALSE
ServerSoftware=Motsognir
ServerSoftwareVersion=1.0
ServerArchitecture=Linux/i386
ServerDescription=This is my gopher server
ServerGeolocationString=Dobrogoszcz, Poland
```

Serving different content on multiple domain names

You have your gopher server up and running, and now you'd like to make use of several different domain names on it – and on each domain, different content should be served. In the http realm such thing is called “virtual hosting”. The gopher protocol, however, doesn't have any provision for a similar mechanism. This doesn't mean that all hope is lost, though.

To work out the above situation, you will need to have multiple different IP addresses assigned to your gopher server (fortunately since IPv6 stepped in, having as many addresses as we need is not a problem anymore). Once this is done, then it's only a matter of 'binding' every one of your gopher domain names to a different IP address. The final step is running a dedicated instance of Motsognir on every IP address.

Example: Let's imagine that I own two domains: gopher.example.com and gopher.mydomain.net. For these domains, I'd like to serve gopher content from /srv/gopher.example.com/ and /srv/gopher.mydomain.net/, respectively.

Step 1: configure at least two different IP addresses on my gopher server, and declare them within my DNS zone to be used like this:

2001:DB8:410E:ABC::1 → gopher.example.com

2001:DB8:410E:ABC::2 → gopher.mydomain.net

Step 2: Prepare two different configuration files for Motsognir, with different “bind” and “GopherHostname” settings, like this:

/etc/motsognir-gopher.example.com.conf

```
GopherHostname=gopher.example.com
bind=2001:DB8:410E:ABC::1
...
```

/etc/motsognir-gopher.mydomain.net.conf

```
GopherHostname=gopher.mydomain.net
bind=2001:DB8:410E:ABC::2
...
```

Step 3: Run two instances of Motsognir, using your custom configuration files:

```
motsognir --config /etc/motsognir-gopher.example.com.conf
motsognir --config /etc/motsognir-gopher.mydomain.net.conf
```

Plugin development

There might be very specific needs that require to add some custom logic into Motsognir. Such needs can be addressed by integrating a 'plugin' into your Motsognir installation. Basically, a plugin is a piece of your own code that can take over any queries that Motsognir receives, and answer to them in lieu of the usual Motsognir processing.

A plugin can be either an executable program or a PHP script. When configured, the plugin application will be called by Motsognir for every incoming gopher query. The unprocessed query is provided to the plugin via the QUERY_STRING environment variable (read the chapter about CGI for other interesting variables). Then, the plugin must decide whether it wants to process the query or not. If not, then it must exit without outputing anything to stdout – Motsognir will process the query as usual then. Otherwise, if the plugin outputs anything to stdout, Motsognir will relay this to the remote client as-is.

The practical uses of such plugin are unlimited. One could image using a custom plugin for statistical accounting, access-list limitations, or even implementation of protocol extensions (think 'gopher+').

Queries filtering

If your plugin is meant to process only a particular kind of queries, then it might be interesting to instruct Motsognir so it relays only these queries to the plugin, to avoid calling the plugin for other queries. This is especially interesting for performance reasons: no need to call the plugin if it's easy to know that it won't be interested in the query anyway. Such configuration is achieved by using the 'PluginFilter' configuration token. PluginFilter is a regular expression that every incoming query will be compared to. Only matching queries are submitted to the plugin, others are directly processed by Motsognir. The PluginFilter regex must be written as a 'POSIX extended' regular expression (same as used by grep -e), as defined by the IEEE Std 1003.1-2001, chapt. 9, Regular Expressions.

Configuration example

The configuration example below would submit requests for all *.jpg and *.png files in the /images path to the /var/myplugin.php plugin:

```
Plugin=/var/myplugin.php
PluginFilter=^/images/.*\.(jpg|png)$
```

Security considerations

Like with any unix daemon, there are a few security aspects that one should always keep in mind. Even the most carefully written programs can have bugs, some of which could be exploited by malicious persons. This is the reason why a system administrator should apply some security limitations even to simplest or most trusted daemons.

Run Motsognir as a non-privileged (non-root) user

A system daemon is usually not supposed to be run as root. In the (unlikely!) situation where an attacker would gain control over Motsognir, having the process running as an unprivileged user would greatly reduce the panel of harmful actions that could be performed on your server. However, there is a problem: you will usually want to run your gopher server under the standard TCP/70 port...and this being a low port requires the process to have root privileges. That's why Motsognir provides a special configuration option called 'RunAsUser'. This option allows to set the username we'd like Motsognir to use, and then, when Motsognir will be launched, it will first open (“bind”) the listening port, and only then drop its privileges to switch to the configured user.

Choose your file permissions wisely

If your Motsognir server runs as a non-privileged user, then it makes much sense to limit permissions on files that it serves. If the gopher server is not supposed to modify a file, this file should be set as 'read-only' and owned by root. This way, even if Motsognir becomes compromised, it still won't be able to modify these files.

Use paranoid mode if you are (really) paranoid

By default, Motsognir will happily serve anything that is located somewhere in the gopher root path, and that is readable by the gopher-running user. To make things even more strict, you might want to use Motsognir's “Paranoid mode” (configurable in Motsognir's configuration file). In this mode, Motsognir will accept to serve only files that have “world readable” permissions set.

Trap the daemon inside a chroot jail

The principle of a chroot jail is simple: run a process inside a 'virtualized' environment with a modified root path (for example, mapping a chroot / on /srv/gopher/). This technique is used to make it impossible for the process to access any file outside the chroot jail. A process can need some files, like shared libraries, or configuration files, to run properly. If chrooting a process, one would need to put all these files into the chroot as well (and the process' executable file itself!). To avoid these problems, Motsognir provides a 'chroot' configuration parameter that, once set, will make Motsognir run, load its configuration, and only then perform a chroot to the designated directory.

Note, that if you use any kind of dynamic files (*.cgi or *.php), you will need to take care to put all dependencies of these applications inside the chroot jail, too. This includes a shell at /bin/sh, all system libs that your applications might require, etc. Often a working /proc will also be needed.

Frequently asked questions (FAQ)

Q: Does Motsognir support special (nationalized) character sets in file names?

A: Yes, it does. Motsognir implements support for UTF-8 encoded URLs, therefore it is able to handle any existing language. Note, that it requires the local server's filesystem to be using UTF-8 too, otherwise only the basic ASCII set will be handled.

Q: Where does Motsognir write its logs?

A: Shortly said, Motsognir does not "write" any logs to your hard disk. Instead, it sends its logs through standard system syslog() calls, and then the syslog subsystem used by your operating system may or may not write these logs somewhere. See your operating system's documentation for details.

Q: Is there any way to run server-side applications on Motsognir?

A: Motsognir supports executable CGI scripts, as well as PHP files. Both of these technologies allow you to run custom server-side scripts.

Q: What's the maximum file size that Motsognir can serve?

A: Motsognir itself can serve files which are up to 8 exbibytes (EiB) big. However, chances are that your filesystem will limit you much sooner (for example EXT3 supports files up to 2 TiB of size, while EXT4 supports files up to 16 TiB).

Q: What does "Motsognir" stand for?

A: In Norse mythology, Mótsognir is the father of the Dwarves. Mótsognir is the creation of Odin and his brothers, Vili and Vé, who fashioned him out of Ymir's blood and bones in the form of a maggot. He got a roughly humanoid appearance and a human-like intelligence, which the rest of the Dwarves later inherited. (source: wikipedia)

Q: Does Motsognir support the HTTP protocol?

A: No. Motsognir is a gopher server. Gopher is a protocol different from HTTP. However, if you send by mistake a HTTP request to Motsognir (for example using a URL like http://yourserver:70/), he will politely answer with a HTTP explanation message.

Q: Is this a real 'FAQ' ?

A: No, I totally made up most of these questions.

Legal mumbo-jumbo

http://motsognir.sourceforge.net

Graphic images

The image of the dwarf on the cover of this manual is based on the original work of Lorenz Frølich (1820-1908).

Trademarks

Unix is a registered trademark of UNIX System Laboratories, Inc. Windows, WindowsNT, and Win32 are registered trademarks of Microsoft Corp. All other product names mentioned herein are the trademarks of their respective owners.