August 17, 2006

Creating a Gopher server with PHP and InetD

This tutorial will teach you how to create a Gopher Server using InetD with PHP. This will teach you how to create a simple socket server using InetD and it will teach you something about the gopher protocol.

Gopher

A long time ago, in the early nineties Gopher was the prefered way to access internet content.. Only later on Tim Berners Lee's HTTP/WWW idea took off. Sixapart recently wrote an article about this chapter of ancient internet history.

Gopher basically has a few main functions and it is kind of restricted to that. This is listing directories (or menu's), serving files and searching. One of the most innovative ideas was that hyperlinks were tightly integrated in the protocol. The modern internet is in a sense based on this important concept. If you want to see a gopher server in action, check out: gopher://gopher.quux.org/. Only a few browser support this protocol, among them are Firefox 1.5 or higher, Camino 1.0 or higher, or a recent Seamonkey or Flock. IE used to support it, but because of a security bug a while ago they didn't fix it, but disabled it instead. Lynx will also work if you're on linux.

InetD

InetD has to be the easiest way to write a socket server, you simply make an entry in /etc/inetd.conf and you can make it work with standard input/output.. more about this later.

What do you need?

The examples here are written for PHP 5.1.x, it will most likely also work in PHP 5.0.x, but you can't run it in 4.x.. further, you need root access to a unix server with inetd installed (usually any linux server comes with inetd). You also need a gopher-compatible browser. You can download the files for this tutorial here, but this is not required.

Lets get started..

We will first write a basic telnet server, because this is the easiest thing to do. We will

make a telnet server that waits for a line to be entered and it will then echo exactly that and break the connection.

If we would be using the socket_* functions things would probably be a bit harder, but for us the script will actually look like this:

```
#!/usr/bin/php
<?php

  $data = fgets(STDIN);
  echo('You said: ' . $data);

?>
```

The first line (#!/usr/bin/php) is a special way of telling linux to use the php interpreter for the rest of the file. If PHP is installed in a non-standard location (other than /usr/bin) you can find it by typing **whereis php** on the command line. fgets() is php's function to return 1 line from an open file. **STDIN** is a constant that refers to unix standard input.

Making it run

Give this testfile the +x permission, you can do that using the following command line:

```
chmod +x filename.php
```

This tells linux (or mac) this file is allowed to run as an executable.

Try running it with `./testsocket.php` (or whatever filename you gave it. If everything worked as expected it will wait till you type something in and press enter, and it will reply with the exact same thing and exit.

Turning it into a telnet server

To do this you will need to edit your **/etc/inetd.conf** file. Open it with your favourite editor and add this line:

```
telnet  stream  tcp     nowait  www-data    /path/to/your/testsocket.php
```

The spaces in between are either regular spaces or tabs. Be sure to change the last parameter to the correct path to your script. This runs the script with the username 'www-data', which is the default username for the apache server on Debian. You might want to change it to the user your apache server runs on.. this can either be www-data, nobody, apache, httpd or a few others. You can also run the script as root, but this gives

the script privileges you might not want to give it (all of them).

This binds the script to the default telnet port (which is 23). Now you need to restart inetd to force it to reload the settings files. You can do that by running the following command (as root):

```
killall -HUP inetd
```

You can try your new telnet server out by running **telnet localhost** from the command line, or if you want to try it from another machine, run : `telnet yourhostname.com` or `telnet://yourhostname.com`.

If it didn't work for you, you might want to check out your system logs.. it could tell you a bit more. You can do that (on some/most) systems with:

```
tail -f /var/log/syslog
```

The last entry(ies) should give you information.

And now.. gopher

In order to do this, I need to explain a bit more about the gopher protocol. When a gopher client connects to a gopher server, it will first send a string containing the information it wants followed by a linebreak ("\n") after that the gopher server throws back the information it requested.

If you go to the root of a gopher server it will start out with a directory listing, like gopher://gopher.quux.org/.

View source in firefox won't help you, gopher uses a special format to submit this directory listing. Every item (including text-lines) are sent on 1 line (separated by \n). This is how a line is built up:

```
[itemType]Name[tab]location[tab]server[tab]port[linebreak]
```

ItemType is a single character which tells the client what type of item this is. 0 means file, 1 means directory, 8 means telnet link, I means an image file and i is informational text.

At the end of the directory listing you'll find a . and the server closes the connection.

A simple gopher server

Add in another line in /etc/inetd.conf. You can write it in the exact same way, but start the line with gopher instead of telnet. Don't forget to restart!

```
gopher  stream  tcp     nowait  www-data    /path/to/your/gopherserver.php
```

For our gopherserver.php, we will start out with a simple class that does some of the work for us. It is *highly* recommended to check out the class first, you can find it here. I decided not to put it here, because there's a lot of code. The code is pretty much self-explanatory.

Now, our gopherserver will look like this:

```
#!/usr/bin/php
<?php

    require_once 'Gopher/Server.php';

    $server = new Gopher_Server();

    $server->setHostname('gopher.rooftopsolutions.nl');

    $server->exec();

?>
```

Now you should have your own gopher server running. The server class is not complete though. Whatever you will serve it, it will reply with the exact same response. If you want plan to make this running you should change the **processRequest** method to return the correct results.

You will find a bunch of constants for the types of files you can serve. If you want to serve a binary file it doesn't matter if you use G_BINARY, G_MACFILE or G_DOSFILE. A bunch of them is not supported in modern clients.

Thats it for this tutorial, we have a proof of concept server running and you should be able to extend it to actually serve information. If people are interested I can make a follow-up tutorial which would explain the search feature, handling urls to the HTTP web.. let me know if it worked for you.