PHP-CLI launches a terminal from QP2SHELL on IBM i

Issue

Interactive calls to a CL program that uses QP2SHELL to run PHP-CLI will result in the end user seeing a PASE terminal display. Even if there is no other need for interaction, the user has to press enter to exit the terminal. This is the expected behavior, but in 32-bit versions of Zend Server this would not happen. Instead, any output from the PHP-CLI call would be sent to a QPRINT spool file. So, for customers upgrading from Zend Server 2019 or earlier to Zend Server 2020 or later, this change in behavior can cause a problem because the end user must now press enter just to continue with the process. This article addresses why 64-bit Zend Server seems to behave differently than 32-bit Zend Server, and offers a quick fix for anyone who would like to avoid a terminal display when calling PHP-CLI interactively from a CL program.

Also fixes cURL issue "getaddrinfo() thread failed to start"

As noted in this article:

getaddrinfo() thread failed to start on IBM i PHP cURL calls in CL programs

Using this shell script with QP2SHELL to call php-cli resolves that issue.

Environment

Any 64-bit version of PHP running on any supported version of IBM i. 64-bit PHP is provided in Zend Server version 2020 and higher, and in any version of ZendPHP running on IBM i.

Details

One difference in the 64-bit compiles of PHP, compared to the 32-bit versions, is that they are compiled in such a way that they do not need to be called from shell scripts (.sh files). In the older versions, the shell script would set up some environmental settings, but now we can just run the binary file directly.

There is a very useful feature of IBM i OS that redirects stdout and stderr output to a spool file when QP2SHELL is called from a batch program. When a binary program is run from a shell script, it runs in a new process, and that new process runs in a batch job. So if QP2SHELL calls a shell script, and that shell script then runs a binary, the output from that binary is created in that new batch job and so is directed to a QPRINT spool file. If the binary is run directly from QP2SHELL, there is no new process and since the job is interactive, the terminal is displayed to show the output. This has always been the case, and this has not changed. What has changed is that php-cli used to be a shell script in the 32-bit versions, but in the 64-bit versions of PHP it is now an executable binary program.

Customers with both 32-bit and 64-bit PHP installed can see this happening pretty easily. So if this explanation seems a little confusing, and you have both versions, try this experiment to help visualize the issue. From a CL command line, with sufficient permissions, run this command:

```
CALL PGM(QP2SHELL) PARM('/usr/local/zendphp7/bin/php-cli' '-v')
```

This is for Zend Server versions 9 through 2019. For earlier versions you would substitute 'zendsvr6' for 'zendphp7'.

When you run this command, it will run briefly and then just finish, with no output displayed. If you want to see the output, you would run this command:

```
WRKSBMJOB *JOB
```

Find the most recent job with the same name as your interactive job, and use option 8 to display its spool files. There will be a QPRINT file that has all the output in it. In this case, it shows the '-v' output from php-cli, which shows the version of PHP and some copyright information.

Now run the same command for the new version of Zend Server, which would be the same except substitute 'zendphp74' for 'zendphp7', like this:

```
CALL PGM(QP2SHELL) PARM('/usr/local/zendphp74/bin/php-cli' '-v')
```

This time the terminal display will appear, and show you the last part of the '-v' output, something like this:

```
- with Zend Java Bridge v2021.0.0, Copyright Zend by Perforce (c) 2021 Pe
rforce Software Inc., by Zend Technologies [loaded] [licensed] [enabled]
 - with Zend Job Queue v2021.0.0, Copyright Zend by Perforce (c) 2021 Perf
orce Software Inc., by Zend Technologies [loaded] [licensed] [enabled]
 - with Zend Utils v2021.0.0, Copyright Zend by Perforce (c) 2021 Perforce
 Software Inc., by Zend Technologies [loaded] [licensed] [enabled]
 - with Zend Code Tracing v2021.0.0, Copyright Zend by Perforce (c) 2021 P
erforce Software Inc., by Zend Technologies [loaded] [licensed] [enabled]
 - with Zend Server Z-Ray v2021.0.0, Copyright Zend by Perforce (c) 2021 P
erforce Software Inc., by Zend Technologies [loaded] [licensed] [enabled]
 - with Zend Monitor v2021.0.0, Copyright Zend by Perforce (c) 2021 Perfor
ce Software Inc., by Zend Technologies [loaded] [licensed] [disabled]
 - with Zend Page Cache v2021.0.0, Copyright Zend by Perforce (c) 2021 Per
force Software Inc., by Zend Technologies [loaded] [licensed] [disabled]
 - with Zend Monitor UI v2021.0.0, Copyright Zend by Perforce (c) 2021 Per
force Software Inc., by Zend Technologies [loaded] [licensed] [enabled]

Press ENTER to end terminal session.
```

You can hit PAGE UP to scroll back to see the first page of the output.

You will notice that if you run more than one command like this in a single session, you will see messages from all the commands whenever you go into the terminal. Those old messages don't just go away when you press enter. You can page up and down to scroll through the terminal output. This can be a nice way to display the output from your PHP script. There is also a response line provided so you can respond to PASE messages, in case there is an error. There are also some function keys you can use to print the output, call up a CL command line, go to the top or bottom, and so on. So this can be a nice option for displaying the output. However, if you prefer to send the output to a spool file, you just need to call php-cli from a shell script. The next section tells how to do it.

Resolution

All you need to do to make things work the way you need them to, is to provide a very simple shell script that you can use to call php-cli. It is just two lines. The first line indicates the shell to use, and the second line just calls php-cli (the name of the php-cli binary is simply "php", although there is a symlink named "php-cli" provided for backward compatibility). Here is the code:

```
#!/bin/sh
/usr/local/zendphp74/php/active/bin/php "$@"
```

Save this code in a script named php-cli.sh. Then just place the attached php-cli.sh script in your /usr/local/zendphp74/bin directory on the IFS. Make sure it has the same permissions as other files in that directory.

You can test it with this command:

```
CALL PGM(QP2SHELL) PARM('/usr/local/zendphp74/bin/php-cli.sh' '-v')
```

This is similar to the command shown in the prior section. But notice we use 'php-cli.sh' instead of 'php-cli' with no '.sh'. When you run it, no terminal should display. Instead, you will find a submitted job that has the output in a QPRINT spool file (WRKSBMJOB *JOB, and look for the most recent job). This is the same as it is currently working in older versions of Zend Server.

Now you should be able to easily change your legacy CLP programs to work with Zend Server 2021. Just change the 'zendphp7' to 'zendphp74', and the 'php-cli' to 'php-cli.sh', in your QP2SHELL calls.

One other thing you might do is put the shell script php-cli.sh in some location other than /usr/local/zendphp74/bin . That directory can get replaced by updates, so custom changes placed in there might be lost in an update. For example, you could create a directory /mydirectory/bin, and put the php-cli.sh file in there. Make sure the permissions match files in /usr/local/zendphp74/bin, and also make sure the permissions for the /mydirectory/bin directory match as well. Then you could change your QP2SHELL calls like this:

```
CALL PGM(QP2SHELL) PARM('/mydirectory/php-cli.sh' '-v')
```

It is a little more work to change the directory names in your CL programs instead of just adding a '4' to 'zendphp7', but it can save you from having to remember to check if you need to reinstall the php-cli.sh script after every update.