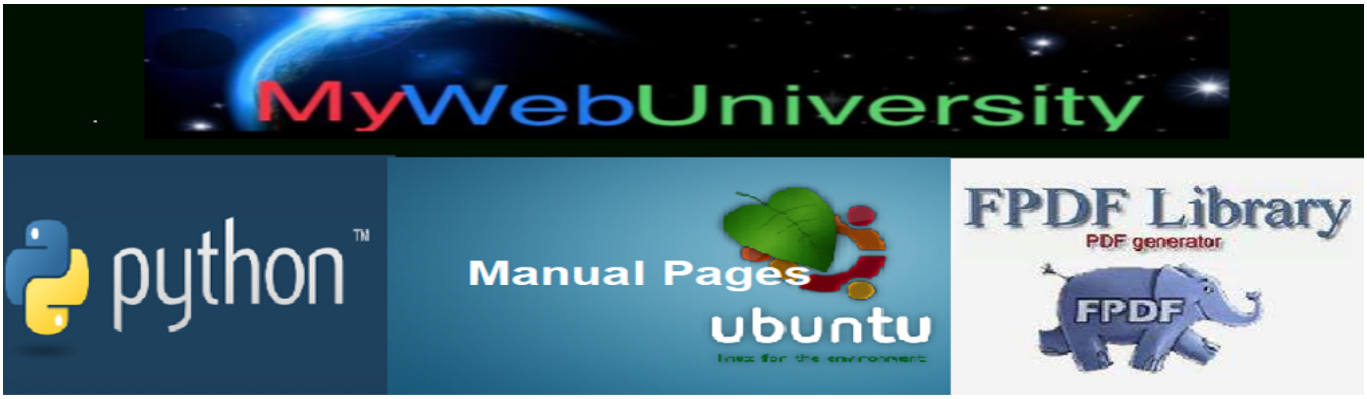

Rocky Enterprise Linux 9.2 Manual Pages on command 'Apache2::SizeLimit.3pm'

$ man Apache2::SizeLimit.3pm

Apache2::SizeLimit(3pm) User Contributed Perl Documentation Apache2::SizeLimit(3pm)

NAME

Apache2::SizeLimit - Because size does matter.

SYNOPSIS

```
PerlLoadModule Apache2::SizeLimit
<Perl>
 Apache2::SizeLimit->set_max_process_size(150_000);   # Max size in KB
 Apache2::SizeLimit->set_min_shared_size(10_000);     # Min share in KB
 Apache2::SizeLimit->set_max_unshared_size(120_000);  # Max unshared size in KB
</Perl>
PerlCleanupHandler Apache2::SizeLimit
```

DESCRIPTION

********** NOIICE **********

This version is only for httpd 2.x and mod_perl 2.x series.

For httpd 1.3.x / mod_perl 1.x Apache::SizeLimit documentation please read the perldoc in lib/Apache/SizeLimit.pm

********** NOTICE **********

This module allows you to kill off Apache httpd processes if they grow too large. You can make the decision to kill a process based on its overall size, by setting a minimum limit on shared memory, or a maximum on unshared memory.

You can set limits for each of these sizes, and if any limit is exceeded, the process will

be killed.

You can also limit the frequency that these sizes are checked so that this module only checks every N requests.

This module is highly platform dependent, please read the "PER-PLATFORM BEHAVIOR" section for details. It is possible that this module simply does not support your platform.

API

You can set set the size limits from a Perl module or script loaded by Apache by calling the appropriate class method on "Apache2::SizeLimit":

- Apache2::SizeLimit->set_max_process_size($size)

  This sets the maximum size of the process, including both shared and unshared memory.

- Apache2::SizeLimit->set_max_unshared_size($size)

  This sets the maximum amount of unshared memory the process can use.

- Apache2::SizeLimit->set_min_shared_size($size)

  This sets the minimum amount of shared memory the process must have.

The two methods related to shared memory size are effectively a no-op if the module cannot determine the shared memory size for your platform. See "PER-PLATFORM BEHAVIOR" for more details.

Running the handler()

There are several ways to make this module actually run the code to kill a process.

The simplest is to make "Apache2::SizeLimit" a "PerlCleanupHandler" in your Apache config:

```
PerlCleanupHandler Apache2::SizeLimit
```

This will ensure that "Apache2::SizeLimit->handler()" is run for all requests.

If you want to combine this module with a cleanup handler of your own, make sure that "Apache2::SizeLimit" is the last handler run:

```
PerlCleanupHandler  Apache2::SizeLimit My::CleanupHandler
```

Remember, mod_perl will run stacked handlers from right to left, as they're defined in your configuration.

If you have some cleanup code you need to run, but stacked handlers aren't appropriate for your setup, you can also explicitly call the "Apache2::SizeLimit->handler()" function from your own cleanup handler:

```
package My::CleanupHandler
sub handler {
    my $r = shift;
```

```
    # Causes File::Temp to remove any temp dirs created during the
    # request
    File::Temp::cleanup();
    return Apache2::SizeLimit->handler($r);
}
```

? Apache2::SizeLimit->add_cleanup_handler($r)

You can call this method inside a request to run "Apache2::SizeLimit"'s "handler()" method for just that request. It's safe to call this method repeatedly -- the cleanup will only be run once per request.

Checking Every N Requests

Since checking the process size can take a few system calls on some platforms (e.g. linux), you may not want to check the process size for every request.

? Apache2::SizeLimit->set_check_interval($interval)

Calling this causes "Apache2::SizeLimit" to only check the process size every $interval requests. If you want this to affect all processes, make sure to call this during server startup.

SHARED MEMORY OPTIONS

In addition to simply checking the total size of a process, this module can factor in how much of the memory used by the process is actually being shared by copy-on-write. If you don't understand how memory is shared in this way, take a look at the mod_perl docs at http://perl.apache.org/docs/.

You can take advantage of the shared memory information by setting a minimum shared size and/or a maximum unshared size. Experience on one heavily trafficked mod_perl site showed that setting maximum unshared size and leaving the others unset is the most effective policy. This is because it only kills off processes that are truly using too much physical RAM, allowing most processes to live longer and reducing the process churn rate.

PER-PLATFORM BEHAVIOR

This module is highly platform dependent, since finding the size of a process is different for each OS, and some platforms may not be supported. In particular, the limits on minimum shared memory and maximum shared memory are currently only supported on Linux and BSD. If you can contribute support for another OS, patches are very welcome.

Currently supported OSes:

linux

For linux we read the process size out of /proc/self/statm. If you are worried about performance, you can consider using "Apache2::SizeLimit->set_check_interval()" to reduce how often this read happens.

As of linux 2.6, /proc/self/statm does not report the amount of memory shared by the copy-on-write mechanism as shared memory. This means that decisions made based on shared memory as reported by that interface are inherently wrong.

However, as of the 2.6.14 release of the kernel, there is /proc/self/smaps entry for each process. /proc/self/smaps reports various sizes for each memory segment of a process and allows us to count the amount of shared memory correctly.

If "Apache2::SizeLimit" detects a kernel that supports /proc/self/smaps and the "Linux::Smaps" module is installed it will use that module instead of /proc/self/statm.

Reading /proc/self/smaps is expensive compared to /proc/self/statm. It must look at each page table entry of a process. Further, on multiprocessor systems the access is synchronized with spinlocks. Again, you might consider using "Apache2::SizeLimit->set_check_interval()".

Copy-on-write and Shared Memory

The following example shows the effect of copy-on-write:

```
 <Perl>
   require Apache2::SizeLimit;
   package X;
   use strict;
   use Apache2::Const -compile => qw(OK);
   my $x = "a" x (1024*1024);
   sub handler {
     my $r = shift;
     my ($size, $shared) = $Apache2::SizeLimit->_check_size();
     $x =~ tr/a/b/;
     my ($size2, $shared2) = $Apache2::SizeLimit->_check_size();
     $r->content_type('text/plain');
     $r->print("1: size=$size shared=$shared\n");
     $r->print("2: size=$size2 shared=$shared2\n");
     return OK;
   }
```

```
</Perl>
<Location /X>
  SetHandler modperl
  PerlResponseHandler X
</Location>
```

The parent Apache process allocates memory for the string in $x. The "tr"-command then overwrites all "a" with "b" if the handler is called with an argument. This write is done in place, thus, the process size doesn't change. Only $x is not shared anymore by means of copy-on-write between the parent and the child.

If /proc/self/smaps is available curl shows:

```
r2@s93:~/work/mp2> curl http://localhost:8181/X
1: size=13452 shared=7456
2: size=13452 shared=6432
```

Shared memory has lost 1024 kB. The process' overall size remains unchanged.

Without /proc/self/smaps it says:

```
r2@s93:~/work/mp2> curl http://localhost:8181/X
1: size=13052 shared=3628
2: size=13052 shared=3636
```

One can see the kernel lies about the shared memory. It simply doesn't count copy-on-write pages as shared.

solaris 2.6 and above

For solaris we simply retrieve the size of /proc/self/as, which contains the address-space image of the process, and convert to KB. Shared memory calculations are not supported.

NOTE: This is only known to work for solaris 2.6 and above. Evidently the /proc filesystem has changed between 2.5.1 and 2.6. Can anyone confirm or deny?

BSD (and OSX)

Uses "BSD::Resource::getrusage()" to determine process size. This is pretty efficient (a lot more efficient than reading it from the /proc fs anyway).

According to recent tests on OSX (July, 2006), "BSD::Resource" simply reports zero for process and shared size on that platform, so OSX is not supported by "Apache2::SizeLimit".

AIX?

Uses "BSD::Resource::getrusage()" to determine process size. Not sure if the shared memory calculations will work or not. AIX users?

Win32

Uses "Win32::API" to access process memory information. "Win32::API" can be installed under ActiveState perl using the supplied ppm utility.

Everything Else

If your platform is not supported, then please send a patch to check the process size. The more portable/efficient/correct the solution the better, of course.

ABOUT THIS MODULE

This module was written in response to questions on the mod_perl mailing list on how to tell the httpd process to exit if it gets too big.

Actually, there are two big reasons your httpd children will grow. First, your code could have a bug that causes the process to increase in size very quickly. Second, you could just be doing operations that require a lot of memory for each request. Since Perl does not give memory back to the system after using it, the process size can grow quite large.

This module will not really help you with the first problem. For that you should probably look into "Apache::Resource" or some other means of setting a limit on the data size of your program. BSD-ish systems have "setrlimit()", which will kill your memory gobbling processes. However, it is a little violent, terminating your process in mid-request.

This module attempts to solve the second situation, where your process slowly grows over time. It checks memory usage after every request, and if it exceeds a threshold, exits gracefully.

By using this module, you should be able to discontinue using the Apache configuration directive MaxRequestsPerChild, although for some folks, using both in combination does the job.

DEPRECATED APIS

Previous versions of this module documented three globals for defining memory size limits:

- $Apache2::SizeLimit::MAX_PROCESS_SIZE
- $Apache2::SizeLimit::MIN_SHARE_SIZE
- $Apache2::SizeLimit::MAX_UNSHARED_SIZE
- $Apache2::SizeLimit::CHECK_EVERY_N_REQUESTS
- $Apache2::SizeLimit::USE_SMAPS

Direct use of these globals is deprecated, but will continue to work for the foreseeable future.

It also documented three functions for use from registry scripts:

- Apache2::SizeLimit::setmax()
- Apache2::SizeLimit::setmin()
- Apache2::SizeLimit::setmax_unshared()

Besides setting the appropriate limit, these functions also add a cleanup handler to the current request. In the 2.x series of mod_perl to use the deprecated functions, you must set PerlOptions +GlobalRequest accordingly.

SUPPORT

The Apache-SizeLimit project is co-maintained by several developers, who take turns at making CPAN releases. Therefore you may find several CPAN directories containing Apache-SizeLimit releases. The best way to find the latest release is to use http://search.cpan.org/.

If you have a question or you want to submit a bug report or make a contribution, please do not email individual authors, but send an email to the modperl <at> perl.apache.org mailing list. This list is moderated, so unless you are subscribed to it, your message will have to be approved first by a moderator. Therefore please allow some time (up to a few days) for your post to propagate to the list.

AUTHOR

Doug Bagley <doug+modperl@bagley.org>, channeling Procrustes.

Brian Moseley <ix@maz.org>: Solaris 2.6 support

Doug Steinwand and Perrin Harkins <perrin@elem.com>: added support for shared memory and additional diagnostic info

Matt Phillips <mphillips@virage.com> and Mohamed Hendawi <mhendawi@virage.com>: Win32 support

Dave Rolsky <autarch@urth.org>, maintenance and fixes outside of mod_perl tree (0.9+).