1 Apache::SizeLimit - Because size does matter.

1.1 Synopsis

This module allows you to kill off Apache httpd processes if they grow too large. You can choose to set up the process size limiter to check the process size on every request:

```
# in your startup.pl:
use Apache::SizeLimit;
# sizes are in KB
$Apache::SizeLimit::MAX_PROCESS_SIZE  = 10000; # 10MB
$Apache::SizeLimit::MIN_SHARE_SIZE    = 1000;  # 1MB
$Apache::SizeLimit::MAX_UNSHARED_SIZE = 12000; # 12MB

# in your httpd.conf:
PerlFixupHandler Apache::SizeLimit
# you can set this up as any Perl*Handler that handles part of the
# request, even the LogHandler will do.
```

Or you can just check those requests that are likely to get big, such as CGI requests. This way of checking is also easier for those who are mostly just running CGI.pm/Registry scripts:

```
# in your CGI:
use Apache::SizeLimit;
&Apache::SizeLimit::setmax(10000);        # Max size in KB
&Apache::SizeLimit::setmin(1000);        # Min share in KB
&Apache::SizeLimit::setmax_unshared(12000); # Max unshared size in KB
```

Since checking the process size can take a few system calls on some platforms (e.g. linux), you may want to only check the process size every *N* times. To do so, put this in your *startup.pl* or CGI:

```
$Apache::SizeLimit::CHECK_EVERY_N_REQUESTS = 2;
```

This will only check the process size every other time the process size checker is called.

1.2 Description

This module allows you to kill off Apache httpd processes if they grow too large.

This module is highly platform dependent, please read the Caveats section.

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, it could have a bug that causes the process to increase in size dramatically, until your system starts swapping. Second, your process just does stuff that requires a lot of memory, and the more different kinds of requests your server handles, the larger the httpd processes grow over time.

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 croak 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. The idea is to check the memory usage after every request, and if it exceeds a threshold, exit 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. Personally, I just use the technique shown in this module and set my `MaxRequestsPerChild` value to 6000.

1.3 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 Sharing Memory section.

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.

1.4 Caveats

This module is platform dependent, since finding the size of a process is pretty different from OS to 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, please do.

Currently supported OSes:

- **linux**

  For linux we read the process size out of */proc/self/status*. This is a little slow, but usually not too bad. If you are worried about performance, try only setting up the the exit handler inside CGIs (with the `setmax` function), and see if the `CHECK_EVERY_N_REQUESTS` option is of benefit.

- **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***

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

- **AIX?**

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

If your platform is not supported, and if you can tell me how to check for the size of a process under your OS (in KB), then I will add it to the list. The more portable/efficient the solution, the better, of course.

1.5 Todo

Possibly provide a perl make/install so that the SizeLimit.pm is created at build time with only the code you need on your platform.

If Apache was started in non-forking mode, should hitting the size limit cause the process to exit?

1.6 Maintainers

Maintainer is the person(s) you should contact with updates, corrections and patches.

- **The documentation mailing list**

1.7 Authors

- Doug Bagley <doug+modperl (at) bagley.org>, channeling Procrustes.
- Brian Moseley <ix (at) maz.org>: Solaris 2.6 support
- Doug Steinwand and Perrin Harkins <perrin (at) elem.com>: added support for shared memory and additional diagnostic info

Only the major authors are listed above. For contributors see the Changes file.

Table of Contents: