1 Apache2::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, or a <Perl> section:
use Apache2::SizeLimit;
# sizes are in KB
$Apache2::SizeLimit::MAX_PROCESS_SIZE  = 12000; # 12MB
$Apache2::SizeLimit::MIN_SHARE_SIZE    = 6000;  # 6MB
$Apache2::SizeLimit::MAX_UNSHARED_SIZE = 5000;  # 5MB

# in your httpd.conf:
PerlCleanupHandler Apache2::SizeLimit
```

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 scripts under `ModPerl::Registry`:

```
# in your script:
use Apache2::SizeLimit;
# sizes are in KB
Apache2::SizeLimit::setmax(12000);
Apache2::SizeLimit::setmin(6000);
Apache2::SizeLimit::setmax_unshared(5000);
```

This will work in places where you are using `SetHandler perl-script` or anywhere you enable `PerlOptions +GlobalRequest`. If you want to avoid turning on `GlobalRequest`, you can pass an `Apache2::RequestRec` object as the second argument in these subs:

```
my $r = shift; # if you don't have $r already
Apache2::SizeLimit::setmax(12000, $r);
Apache2::SizeLimit::setmin(6000, $r);
Apache2::SizeLimit::setmax_unshared(5000, $r);
```

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:

```
$Apache2::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 is highly platform dependent, please read the Caveats section. It also does not work under threaded MPMs.

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, it may just do things 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 `Apache2::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 you can use both if you are feeling paranoid. Most users use the technique shown in this module and set their `MaxRequestsPerChild` value to `0`.

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 extensive documentation 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.

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.

1.4.1 Supported OSes

- **linux**

  For linux we read the process size out of */proc/self/statm*. This seems to be fast enough on modern systems. If you are worried about performance, try setting the `CHECK_EVERY_N_REQUESTS` option.

Since linux 2.6 */proc/self/statm* does not report the amount of memory shared by the copy-on-write mechanism as shared memory. Hence decisions made on the basis of `MAX_UNSHARED_SIZE` or `MIN_SHARE_SIZE` are inherently wrong.

To correct the situation there is a patch to the linux kernel that adds a */proc/self/smaps* entry for each process. At the time of this writing the patch is included in the mm-tree (linux-2.6.13-rc4-mm1) and is expected to make it into the vanilla kernel in the near future.

/proc/self/smaps reports various sizes for each memory segment of a process and allows to count the amount of shared memory correctly.

If `Apache2::SizeLimit` detects a kernel that supports */proc/self/smaps* and if the `Linux::Smaps` module is installed it will use them instead of */proc/self/statm*. You can prevent `Apache2::SizeLimit` from using */proc/self/smaps* and turn on the old behaviour by setting `$Apache2::SizeLimit::USE_SMAPS` to 0 before the first check.

`Apache2::SizeLimit` also resets `$Apache2::SizeLimit::USE_SMAPS` to 0 if it somehow decides not to use */proc/self/smaps*. Thus, you can check it to determine what is actually used.

NOTE: 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 spin-locks. Hence, you are encouraged to set the `CHECK_EVERY_N_REQUESTS` option.

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

```
<Perl>
  require Apache2::SizeLimit;
  package X;
  use strict;
  use Apache2::RequestRec ();
  use Apache2::RequestIO ();
  use Apache2::Const -compile=>qw(OK);

  my $x= "a" x (1024*1024);

  sub handler {
    my $r = shift;
    my ($size, $shared) = $Apache2::SizeLimit::HOW_BIG_IS_IT->();
    $x =~ tr/a/b/;
    my ($size2, $shared2) = $Apache2::SizeLimit::HOW_BIG_IS_IT->();
    $r->content_type('text/plain');
    $r->print("1: size=$size shared=$shared\n");
    $r->print("2: size=$size2 shared=$shared2\n");
    return Apache2::Const::OK;
  }
</Perl>

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

The parent apache allocates a megabyte 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**

  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?

- **Win32**

  Under mod_perl 1, SizeLimit provided basic functionality by using `Win32::API` to access process memory information. This worked because there was only one mod_perl thread. With mod_perl 2, Win32 runs a true threaded MPM, which unfortunately means that we can't tell the size of each interpreter. Win32 support is disabled until a solution for this can be found.

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

1.4.2 Supported MPMs

At this time, `Apache2::SizeLimit` does not support use under threaded MPMs. This is because there is no efficient way to get the memory usage of a thread, or make a thread exit cleanly. Suggestions and patches are welcome on the mod_perl dev mailing list.

1.5 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 2.0.

1.6 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

Torsten Foertsch <torsten.foertsch gmx.net>: Linux::Smaps support

Table of Contents:

1 Apache2::SizeLimit - Because size does matter. 1
1.1 Synopsis 2
1.2 Description 2
1.3 Shared Memory Options 3
1.4 Caveats 3
1.4.1 Supported OSes 3
1.4.2 Supported MPMs 6
1.5 Copyright 6
1.6 Author 6