1 Troubleshooting mod_perl problems

1.1 Description

Frequently encountered problems (warnings and fatal errors) and their troubleshooting.

1.2 Building and Installation

1.2.1 Cannot find -lgdbm / libgdbm.so.3: open failed: No such file or directory

Please see: Missing or Misconfigured libgdbm.so.

Also it seems that on Solaris this exact issue doesn't show up at compile time, but at run time, so you may see the errors like:

```
.../mod_perl-1.99_17/blib/arch/auto/APR/APR.so' for module APR:
ld.so.1: /usr/local/ActivePerl-5.8/bin/perl: fatal:
libgdbm.so.3: open failed: No such file or directory at
...5.8.3/sun4-solaris-thread-multi/DynaLoader.pm line 229.
```

the solution is the same, make sure that you have the libgdbm shared library and it's properly symlinked.

1.3 Configuration and Startup

1.3.1 Can't locate TestFilter/in_str_consume.pm in @INC...

Sometimes you get a problem of perl not being able to locate a certain Perl module. This can happen in the mod_perl test suite or in the normal mod_perl setup. One of the possible reasons is a low limit on the number of files that can be opened by a single process. To check whether this is the problem run the process under `strace(1)` or an equivalent utility.

For example on OpenBSD 3.5 the default setting for a maximum number of files opened by a single process seems to be 64, so when you try to run the mod_perl test suite, which opens a few hundreds of files, you will have a problem. e.g. the test suite may fail as:

```
[Wed Aug 25 09:49:40 2004] [info] 26 Apache2:: modules loaded
[Wed Aug 25 09:49:40 2004] [info] 7 APR:: modules loaded
[Wed Aug 25 09:49:40 2004] [info] base server + 20 vhosts ready
to run tests
[Wed Aug 25 09:49:40 2004] [error] Can't locate
TestFilter/in_str_consume.pm in @INC (@INC contains: ...
```

Running the system calls tracing program (`ktrace(1)` on OpenBSD, `strace(1)` on Linux):

```
% sudo ktrace -d /usr/local/apache/bin/httpd -d /tmp/mod_perl-2.0/t \
  -f /tmp/mod_perl-2.0/t/conf/httpd.conf -DAPACHE2 -X
```

looking at the ktrace dump reveals:

```
16641 httpd    NAMI "/tmp/mod_perl-2.0/t/lib/TestFilter/in_str_consume.pmc"
16641 httpd    RET   stat -1 errno 2 No such file or directory
16641 httpd    CALL  open(0x3cdae100,0,0)
16641 httpd    RET   open -1 errno 24 Too many open files
```

It's clear that Perl can't load *TestFilter/in_str_consume.pm* because it can't open the file.

This problem can be resolved by increasing the open file limit to 128 (or higher):

```
$ ulimit -n 128
```

1.3.2 "mod_perl.c" is not compatible with this version of Apache (found 20020628, need 20020903)

That error message means that mod_perl was built against Apache released on or post-20020628, but you are trying to load it against one released on or post-20020903. You will see the same error message for any other Apache module -- this is an error coming from Apache, not mod_perl.

Apache bumps up a special magic number every time it does a binary incompatible change, and then it makes sure that all modules that it loads were compiled against the same compatibility generation (which may include only one or quite a few Apache releases).

You may encounter this situation when you upgrade to a newer Apache, without rebuilding mod_perl. Or when you have several versions of Apache installed on the same system. Or when you install prepackaged binary versions which aren't coming from the source and aren't made against the same Apache version.

The solution is to have mod_perl built against the same Apache installed on your system. So either build from source or contact your binary version supplier and get a proper package(s) from them.

1.3.3 Server Hanging at the Startup

First you need to figure out where it hangs. strace(1) or an equivalent utility can be used to discover which call the server hangs on. You need to start the process in the single server mode so you will have only one process to monitor.

For example if the server hangs during 'make test', you should run:

```
% cd modperl-2.0
% strace /path/to/httpd -d t -f t/conf/httpd.conf \
  -DAPACHE2 -DONE_PROCESS -DNO_DETATCH
```

(and may be `-DPERL_USEITHREADS` if it was in the original output of `make test`.)

If the trace ends with:

```
open("/dev/random", O_RDONLY)           = 3
read(3, <unfinished ...>
```

then you have a problem with your OS, as */dev/random* doesn't have enough entropy to give the required random data, and therefore it hangs. This may happen in `apr_uuid_get()` C call or Perl `APR::UUID->new`.

The solution in this case is either to fix the problem with your OS, so that

```
% perl -le 'open I, "/dev/random"; read I, $d, 10; print $d'
```

will print some random data and not block. Or you can use an even simpler test:

```
% cat /dev/random
```

which should print some random data and not block.

If you can't fix the OS problem, you can rebuild Apache 2.0 with `--with-devrandom=/dev/urandom` - however, that is not secure for certain needs. Alternatively setup EGD and rebuild Apache 2.0 with `--with-egd`. Apache 2.1/apr-1.1 will have a self-contained PRNG generator built-in, which won't rely on */dev/random*.

1.3.4 (28)No space left on device

httpd-2.0 is not very helpful at telling which device has run out of precious space. Most of the time when you get an error like:

```
(28)No space left on device:
mod_rewrite: could not create rewrite_log_lock
```

it means that your system have run out of semaphore arrays. Sometimes it's full with legitimate semaphores at other times it's because some application has leaked semaphores and haven't cleaned them up during the shutdown (which is usually the case when an application segfaults).

Use the relevant application to list the ipc facilities usage. On most Unix platforms this is usually an `ipcs(1)` utility. For example linux to list the semaphore arrays you should execute:

```
% ipcs -s
------ Semaphore Arrays --------
key        semid      owner      perms      nsems
0x00000000 2686976    stas      600        1
0x00000000 2719745    stas      600        1
0x00000000 2752514    stas      600        1
```

Next you have to figure out what are the dead ones and remove them. For example to remove the semid 2719745 execute:

```
% ipcrm -s 2719745
```

Instead of manually removing each (and sometimes there can be many of them), and if you know that none of listed the semaphores is really used (all leaked), you can try to remove them all:

```
% ipcs -s | perl -ane ''ipcrm -s $F[1]''
```

httpd-2.0 seems to use the key `0x00000000` for its semaphores on Linux, so to remove only those that match that key you can use:

```
% ipcs -s | perl -ane '/^0x00000000/ && 'ipcrm -s $F[1]''
```

Notice that on other platforms the output of `ipcs -s` might be different, so you may need to apply a different Perl one-liner.

1.3.5 Segmentation Fault when Using DBI

Update DBI to at least version 1.31.

1.3.6 <Perl> directive missing closing '>'

See the Apache2::PerlSections manpage.

1.3.7 'Invalid per-unknown PerlOption: ParseHeaders' on HP-UX 11 for PA-RISC

When building mod_perl 2.0 on HP-UX 11 for PA-RISC architecture, using the HP ANSI C compiler, please make sure you have installed patches PHSS_29484 and PHSS_29485. Once installed the issue should go away.

1.4 Shutdown and Restart

Issues happening during server shutdown and restart, or during specific interpreter shutdown at runtime with threaded mpm.

1.4.1 Subroutines in <perl> sections under threaded mpm

If you have defined a subroutine inside a <perl> section, under threaded mpm (or under perl with enabled ithreads which spawn its own ithreads), like so:

```
<Perl>
  sub foo {}
</Perl>
```

At the server shutdown, or when any interpreter quits you will see the following error in the *error_log*:

```
Attempt to free temp prematurely: SV 0x91b8e74,
Perl interpreter: 0x8547698 during global destruction.
Scalars leaked: 1
```

This is a bug in Perl and as of Perl 5.8.4 it's not resolved. For more information see:

http://rt.perl.org:80/rt3/Ticket/Display.html?id=29018

1.4.2 Modules using `Scalar::Util::weaken` under threaded mpm

Modules using `Scalar::Util::weaken` under threaded mpm may get:

```
Attempt to free unreferenced scalar SV 0x8154f74.
```

when each interprter exits.

This is a bug in Perl and as of Perl 5.8.4 it's not resolved. For more information see:

http://rt.perl.org:80/rt3/Ticket/Display.html?id=24660

1.5 Code Parsing and Compilation

1.5.1 Segfault with __read_nocancel Backtrace

If your application segfaults and you get a similar to the following backtrace:

```
(gdb) bt
#0  0x4030d4d1 in __read_nocancel () from /lib/tls/libpthread.so.0
#1  0x00000000 in ?? ()
```

that usually means that you've build your non-mod_perl modules with ithreads enabled perl. Then you have built a new perl **without** ithreads. But you didn't nuke/rebuild the old non-mod_perl modules. Now when you try to run those, you get the above segfault. To solve the problem recompile all the modules. The easiest way to accomplish that is to either remove all the modules completely, build the new perl and then install the new modules. You could also try to create a bundle of the existing modules using `CPAN.pm` prior to deleting the old modules, so you can easily reinstall all the modules you previously had.

1.5.2 Registry scripts fail to load with: Unrecognized character \xEF at ...

Certain editors (in particular on win32) may add a UTF-8 Byte Order Marker (BOM: http://www.unicode.org/faq/utf_bom.html#BOM) at the beginning of the file. Since `ModPerl::RegistryCooker` adds extra code in front of the original script, before compiling it, it creates a situation where BOM appears past the beginning of the file, which is why the error:

```
Unrecognized character \xEF at ...
```

is thrown by Perl.

The simplest solution is to configure your editor to not add BOMs (or switch to another editor which allows you to do that).

You could also subclass `ModPerl::RegistryCooker` or its existing subclasses to try to remove BOM in ModPerl::RegistryCooker::read_script():

```
# remove BOM
${$self->{CODE}} =~ s/^(?:
    \xef\xbb\xbf     |
    \xfe\xff         |
    \xff\xfe         |
    \x00\x00\xfe\xff |
    \xff\xfe\x00\x00
)//x;
```

but do you really want to add an overhead of this operation multiple times, when you could just change the source file once? Probably not. It was also reported that on win32 the above s/// doesn't work.

1.6 Runtime

1.6.1 error_log is Full of Escaped \n, \t, etc.

It's an Apache "feature", see `-DAP_UNSAFE_ERROR_LOG_UNESCAPED`.

1.6.2 Problems with Catching Signals

See Using Signal Handlers.

1.6.3 APR::Socket::recv: (11) Resource temporarily unavailable at ...

You need to make sure that the socket is set to blocking IO mode before using it.

1.6.4 `APR::UUID->new` Hanging

See Server Hanging at the Startup.

1.6.5 Memory Leaks

- **s/// in perls 5.8.1 and 5.8.2**

  p5-porters report: http://www.xray.mpe.mpg.de/mailing-lists/perl5-porters/2003-12/msg00634.html

Fixed in 5.8.3. There is no workaround but to upgrade to 5.8.3 or higher.

1.6.6 C Libraries Don’t See %ENV Entries Set by Perl Code

For example some people have reported problems with `DBD::Oracle` (whose guts are implemented in C), which doesn’t see environment variables (like `ORACLE_HOME`, `ORACLE_SID`, etc.) set in the perl script and therefore fails to connect.

The issue is that the C array `environ[]` is not thread-safe. Therefore mod_perl 2.0 unties `%ENV` from the underlying `environ[]` array under the *perl-script* handler.

The `DBD::Oracle` driver or client library uses `getenv()` (which fetches from the `environ[]` array). When `%ENV` is untied from `environ[]`, Perl code will see `%ENV` changes, but C code will not.

The *modperl* handler does not untie `%ENV` from `environ[]`. Still one should avoid setting `%ENV` values whenever possible. And if it is required, should be done at startup time.

In the particular case of the `DBD::` drivers, you can set the variables that don’t change (`$ENV{ORACLE_HOME}` and `$ENV{NLS_LANG}` in the startup file, and those that change pass via the `connect()` method, e.g.:

```
my $sid      = 'ynt0';
my $dsn      = 'dbi:Oracle:';
my $user     = 'username/password';
my $password = '';
$dbh = DBI->connect("$dsn$sid", $user, $password)
    or die "Cannot connect: " . $DBI::errstr;
```

Also remember that `DBD::Oracle` requires that *ORACLE_HOME* (and any other stuff like *NSL_LANG* stuff) be in `%ENV` when `DBD::Oracle` is loaded (which might happen indirectly via the `DBI` module. Therefore you need to make sure that wherever that load happens `%ENV` is properly set by that time.

Another solution that works **only with prefork mpm**, is to use `Env::C` (http://search.cpan.org/dist/Env-C/). This module sets the process level environ, bypassing Perl’s `%ENV`. This module is not thread-safe, due to the nature of environ process struct, so don’t even try using it in a threaded environment.

1.6.7 Error about not finding Apache.pm with CGI.pm

You need to install at least version 3.11 of CGI.pm to work under mod_perl 2.0, as earlier CGI.pm versions aren’t mod_perl 2.0 aware.

1.6.8 20014:Error string not specified yet

This error is reported when some undefined Apache error happens. The known cases are:

- **when using mod_deflate**

  A bug in mod_deflate was triggering this error, when a response handler would flush the data that was flushed earlier: http://nagoya.apache.org/bugzilla/show_bug.cgi?id=22259 It has been fixed in httpd-2.0.48.

1.6.9 (22)Invalid argument: core_output_filter: writing data to the network

Apache uses the sendfile syscall on platforms where it is available in order to speed sending of responses. Unfortunately, on some systems, Apache will detect the presence of sendfile at compile-time, even when it does not work properly. This happens most frequently when using network or other non-standard file-system.

The whole story and the solutions are documented at:
http://httpd.apache.org/docs-2.0/faq/error.html#error.sendfile

1.6.10 undefined symbol: apr_table_compress

After a successful mod_perl build, sometimes during the startup or a runtime you'd get an "undefined symbol: foo" error. The following is one possible scenario to encounter this problem and possible ways to resolve it.

Let's say you ran mod_perl's test suite:

```
% make test
```

and got errors, and you looked in the *error_log* file (*t/logs/error_log*) and saw one or more "undefined symbol" errors, e.g.

```
% undefined symbol: apr_table_compress
```

- **Step 1**

  From the source directory (same place you ran "make test") run:

  ```
  % ldd blib/arch/auto/APR/APR.so | grep apr-
  ```

  ldd is not available on all platforms, e.g. not on Darwin/OS X. Instead on Darwin/OS X, you can use their otool.

  You you should get a full path, for example:

  ```
  libapr-0.so.0 => /usr/local/apache2/lib/libapr-0.so.0 (0x40003000)
  ```

  or

```
libapr-0.so.0 => /some/path/to/apache/lib/libapr-0.so.0 (0x40003000)
```

or something like that. It's that full path to libapr-0.so.0 that you want.

- **Step 2**

  Do:

  ```
  % nm /path/to/your/libapr-0.so.0 | grep table_compress
  ```

  for example:

  ```
  % nm /usr/local/apache2/lib/libapr-0.so.0 | grep table_compress
  ```

  You should get something like this:

  ```
  0000d010 T apr_table_compress
  ```

  If you get the message:

  ```
  nm: /usr/local/apache2/lib/libapr-0.so.0: no symbols
  ```

  that means that the library was stripped. You probably want to obtain Apache 2.x or libapr source, matching your binary and check it instead. Or rebuild it with debugging enabled, which will not strip the symbols.

  Note that the "grep table_compress" is only an example, the exact string you are looking for is the name of the "undefined symbol" from the *error_log* file. So, if you get:

  ```
  undefined symbol apr_holy_grail
  ```

  then you would do:

  ```
  % nm /usr/local/apache2/lib/libapr-0.so.0 | grep holy_grail
  ```

- **Step 3**

  Now, let's see what shared libraries your apache binary has. So, if in step 1 you got */usr/local/apache2/lib/libapr-0.so.0* then you will do:

  ```
  % ldd /usr/local/apache2/bin/httpd
  ```

  if in step 1 you got */foo/bar/apache/lib/libapr-0.so.0* then you do:

  ```
  % ldd /foo/bar/apache/bin/httpd
  ```

  The output should look something like this:

  ```
  libssl.so.2 => /lib/libssl.so.2 (0x40023000)
  libcrypto.so.2 => /lib/libcrypto.so.2 (0x40054000)
  libaprutil-0.so.0 => /usr/local/apache2/lib/libaprutil-0.so.0 (0x40128000)
  libgdbm.so.2 => /usr/lib/libgdbm.so.2 (0x4013c000)
  libdb-4.0.so => /lib/libdb-4.0.so (0x40143000)
  ```

```
libexpat.so.0 => /usr/lib/libexpat.so.0 (0x401eb000)
libapr-0.so.0 => /usr/local/apache2/lib/libapr-0.so.0 (0x4020b000)
librt.so.1 => /lib/librt.so.1 (0x40228000)
libm.so.6 => /lib/i686/libm.so.6 (0x4023a000)
libcrypt.so.1 => /lib/libcrypt.so.1 (0x4025c000)
libnsl.so.1 => /lib/libnsl.so.1 (0x40289000)
libdl.so.2 => /lib/libdl.so.2 (0x4029f000)
libpthread.so.0 => /lib/i686/libpthread.so.0 (0x402a2000)
libc.so.6 => /lib/i686/libc.so.6 (0x42000000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

Those are name => value pairs showing the shared libraries used by the `httpd` binary.

Take note of the value for *libapr-0.so.0* and compare it to what you got in step 1. They should be the same, if not, then mod_perl was compiled pointing to the wrong Apache installation. You should run "make clean" and then

```
% perl Makefile.pl MP_APACHE_CONFIG=/path/to/apache/bin/apr-config
```

using the correct path for the Apache installation.

- **Step 4**

  You should also search for extra copies of *libapr-0.so.0*. If you find one in */usr/lib* or */usr/local/lib* that will explain the problem. Most likely you have an old pre-installed apr package which gets loaded before the copy you found in step 1.

  On most Linux (and Mac OS X) machines you can do a fast search with:

  ```
  % locate libapr-0.so.0
  ```

  which searches a database of files on your machine. The "locate" database isn’t always up-to-date so a slower, more comprehensive search can be run (as root if possible):

  ```
  % find / -name "libapr-0.so.0*"
  ```

  or

  ```
  % find /usr/local -name "libapr-0.so.0*"
  ```

  You might get output like this:

  ```
  /usr/local/apache2.0.47/lib/libapr-0.so.0.9.4
  /usr/local/apache2.0.47/lib/libapr-0.so.0
  /usr/local/apache2.0.45/lib/libapr-0.so.0.9.3
  /usr/local/apache2.0.45/lib/libapr-0.so.0
  ```

  in which case you would want to make sure that you are configuring and compiling mod_perl with the latest version of apache, for example using the above output, you would do:

```
% perl Makefile.PL MP_AP_CONFIG=/usr/local/apache2.0.47
% make
% make test
```

There could be other causes, but this example shows you how to act when you encounter this problem.

1.6.11 Variable $x will not stay shared at

This warning is normally as a result of variables that your script is sharing with subroutines globally, rather than passing by value or reference. As the cause and solution of this is virtually identical to another commonly encountered problem (Sometimes it works, sometimes it doesn't), the text is not repeated here but is instead included in that section which follows this one.

You may have read somewhere *out there* that this warning can be ignored, but if you read on you will see that you should *never* ignore the warning. The other thing that might confuse you is that this warning is normally encountered when defining subroutines within subroutines. So why would you experience it in your script where that is not the case? The reason is because mod_perl wraps your script in its own subroutine (see the Perl Reference documentation for more details).

1.6.12 Sometimes it Works, Sometimes it Doesn't

When you start running your scripts under mod_perl, you might find yourself in a situation where a script seems to work, but sometimes it screws up. And the more it runs without a restart, the more it screws up. Often the problem is easily detectable and solvable. You have to test your script under a server running in single process mode (`httpd -X`).

Generally the problem is the result of using global variables (normally accompanied by a Variable $x will not stay shared at warning). Because global variables don't change from one script invocation to another unless you change them, you can find your scripts do strange things.

Let's look at three real world examples:

1.6.12.1 An Easy Break-in

The first example is amazing: Web Services. Imagine that you enter some site where you have an account, perhaps a free email account. Having read your own mail you decide to take a look at someone else's.

You type in the username you want to peek at and a dummy password and try to enter the account. On some services this will work!!!

You say, why in the world does this happen? The answer is simple: **Global Variables**. You have entered the account of someone who happened to be served by the same server child as you. Because of sloppy programming, a global variable was not reset at the beginning of the program and voila, you can easily peek into someone else's email! Here is an example of sloppy code:

```
use vars ($authenticated);
my $q = new CGI;
my $username = $q->param('username');
my $passwd   = $q->param('passwd');
```

```
authenticate($username,$passwd);
  # failed, break out
unless ($authenticated){
  print "Wrong passwd";
  exit;
}
  # user is OK, fetch user's data
show_user($username);

sub authenticate{
  my ($username,$passwd) = @_;
  # some checking
  $authenticated = 1 if SOME_USER_PASSWD_CHECK_IS_OK;
}
```

Do you see the catch? With the code above, I can type in any valid username and any dummy password and enter that user's account, provided she has successfully entered her account before me using the same child process! Since `$authenticated` is global--if it becomes 1 once, it'll stay 1 for the remainder of the child's life!!! The solution is trivial--reset `$authenticated` to 0 at the beginning of the program.

A cleaner solution of course is not to rely on global variables, but rely on the return value from the function.

```
my $q = CGI->new;
my $username = $q->param('username');
my $passwd   = $q->param('passwd');
my $authenticated = authenticate($username,$passwd);
  # failed, break out
unless ($authenticated){
  print "Wrong passwd";
  exit;
}
  # user is OK, fetch user's data
show_user($username);

sub authenticate{
  my ($username,$passwd) = @_;
  # some checking
  return (SOME_USER_PASSWD_CHECK_IS_OK) ? 1 : 0;
}
```

Of course this example is trivial--but believe me it happens!

1.6.12.2 Thinking mod_cgi

Just another little one liner that can spoil your day, assuming you forgot to reset the `$allowed` variable. It works perfectly OK in plain mod_cgi:

```
$allowed = 1 if $username eq 'admin';
```

But using mod_perl, and if your system administrator with superuser access rights has previously used the system, anybody who is lucky enough to be served later by the same child which served your administrator will happen to gain the same rights.

The obvious fix is:

```
$allowed = $username eq 'admin' ? 1 : 0;
```

1.6.12.3 Regular Expression Memory

Another good example is usage of the `/o` regular expression modifier, which compiles a regular expression once, on its first execution, and never compiles it again. This problem can be difficult to detect, as after restarting the server each request you make will be served by a different child process, and thus the regex pattern for that child will be compiled afresh. Only when you make a request that happens to be served by a child which has already cached the regex will you see the problem. Generally you miss that. When you press reload, you see that it works (with a new, fresh child). Eventually it doesn't, because you get a child that has already cached the regex and won't recompile because of the `/o` modifier.

An example of such a case would be:

```
my $pat = $q->param("keyword");
foreach( @list ) {
  print if /$pat/o;
}
```

To make sure you don't miss these bugs always test your CGI in single process mode.

To solve this particular `/o` modifier problem refer to Compiled Regular Expressions.

For more details and further examples please see the Perl Reference documentation.

1.7 Issues with APR Used Outside of mod_perl

It doesn't strictly belong to this document, since it's talking about APR usages outside of mod_perl, so this may move to its own dedicated page, some time later.

Whenever using an `APR::` package outside of mod_perl, you need to:

```
use APR;
```

in order to load the XS subroutines. For example:

```
% perl -MAPR -MAPR::UUID -le 'print APR::UUID->new->format'
```

1.8 Maintainers

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

- **Stas Bekman**

1.9 Authors

- **Stas Bekman**

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

Table of Contents: