

Apache internals and debugging

Ryan Matteson

matty91@gmail.com

<http://prefetch.net>

Presentation overview

- An introduction to the Apache source code layout
- An introduction to key concepts such as modules, buckets, brigades, hooks, filters, and MPMs
- An introduction to several tools that can be used to perform post-mortem analysis
- An introduction to several tools that can be used to debug transient server problems

Apache layout

What is Apache?

- Flexible and extensible opensource web server
- Supports HTTP versions 0.9, 1.0 and 1.1
- Ships with a variety of modules that can be used to secure communications and transform content
- Supports virtual hosting
- Supports static and dynamic content delivery
- The list goes on and on ...

How big is Apache?

- 150k+ lines of C code*:

```
$ find . -name \*.c -exec egrep -v '([ ]+|^|^$|^*|^*^)* \{ |; | wc -l' 155884
```

- 480+ source code files:

```
$ find . -name '*.c' -ls | wc -l
488
```

- 240+ header files:

```
$ find . -name '*.h' -ls | wc -l
247
```

* Several source code files are used for experimentation, testing, and debugging purposes

How is Apache organized?

- Apache is broken up into several pieces
 - The server core
 - The Apache portable runtime
 - The Apache portable runtime utilities
 - Support infrastructure
 - Numerous modules

How is the Apache source code organized?

- The server core source code resides in \$SRCROOT/server
- The server core header files reside in \$SRCROOT/include
- The portable runtime resides in \$SRCROOT/src/lib/apr
- The portable runtime utilities reside in \$SRCROOT/src/lib/apr-util
- Support infrastructure resides in \$SRCROOT/support
- Modules reside in \$SRCROOT/modules

Apache terminology

Resource pools

- Pools are used to simplify memory management
- Pools are created and destroyed automatically by the server core
- Pools have a lifetime associated with them (e.g, a request pool is created when a request arrives, and destroyed after the request is processed)
- `apr_pools.h` provides a thorough description of the pool API and the underlying allocator

Multi-processing modules (MPMs)

- MPMs are responsible for accepting network requests and dispatching those requests to children (processes or threads)
- Two main MPMs in use today
 - Prefork MPM utilizes processes to handle requests
 - Worker MPM utilizes a threaded multi-process model process requests
- `mpm_common.h`, `prefork.c` and `worker.c` contain the MPM, prefork and worker MPM implementations

Modules and hooks

- Apache uses modules to isolate functionality and to extend the core servers capabilities
- Modules utilize hooks and filters to tie into the request processing flow
- The SHOW_HOOKS environment variable can be used to watch hook processing order

```
$ export SHOW_HOOKS=1 && httpd -X
```

```
Registering hooks for core.c
```

```
Hooked create_connection
```

```
Hooked pre_connection
```

```
...
```

Buckets and brigades

- Buckets are abstractions used by Apache to store data as it's read from or written to the network
- Bucket API provides a rich set of functions to modify bucket contents
- Brigades are chains of buckets used to efficiently pass data between filters
- Brigade API provides a rich set of functions to modify the members in a brigade
- `apr_buckets.h` provides a thorough description of buckets and brigades

Filters

- Filters are used to read and transform data as it's read from and written to the network
- Filter chains are used to allow the output from one filter to become the input to another filter
- Two type of filters
 - Input filters (e.g., `mod_deflate`, `mod_ssl`)
 - Output filters (e.g., `mod_ssl`, `mod_headers`)
- `util_filter.h` provides a thorough description of input and output filter chains

Debugging

Why debug?

- Because software breaks
- Because administrators get frustrated with messages similar to the following:
[Sat Feb 04 13:00:27 2006] [notice] child pid 8581 exit signal Segmentation fault (11), possible coredump in /var/tmp/apache2
- And most importantly, because debugging can be fun and rewarding

Types of debugging

- Post-mortem debugging
- Transient problem debugging
- Reverse engineering

Postmortem debugging

What is post-mortem debugging?

- Post-mortem debugging is the art of finding out why a system failed given a set of determinants (e.g., core file, audit trail, logfile messages)
- Software post-mortem analysis typically relies on custom instrumented binaries, logfile messages, and core file analysis
- Several tools can help with performing Apache post-mortem analysis
 - Apache maintainer mode
 - `mod_backtrace`
 - GDB and the Apache GDB macros

Maintainer mode

- Builds Apache with debugging support and custom instrumentation
- Enabled at configure time

```
$ ./configure --enable-maintainer-mode \  
               --enable-exception-hook \  
               --enable-mods-shared=most \  
               --enable-deflate=shared \  
               --prefix=/opt/apache2
```

mod_backtrace

- Writes a stack trace to a logfile each time a critical signal (e.g., SIGSEGV) is received
- Utilizes the Apache exception hook
- Can help with diagnosing problems on platforms that don't reliably create core files
- Supports Linux and FreeBSD
- Solaris patch available on my website

Building and installing mod_backtrace

- Enable exception hook

```
$ ./configure --enable-exception-hook ...
```

- Compile mod_backtrace module

```
$ apxs -ci mod_backtrace.c
```

- Enable module

```
LoadModule backtrace_module modules/mod_backtrace.so
```

```
EnableExceptionHook On
```

Watch Apache go splat

- Check the error_log when Apache misbehaves

```
$ kill -SIGSEGV `pgrep httpd | tail -1`
```

```
$ tail -100 error_log
```

```
[Sat Feb  4 20:36:05 2006] pid 23514 mod_backtrace backtrace for sig 11 (thread "pid" 23514)
[Sat Feb  4 20:36:05 2006] pid 23514 mod_backtrace main() is at 326b0
/var/tmp/apache2/modules/mod_backtrace.so:bt_exception_hook+0x108
/var/tmp/apache2/bin/httpd:ap_run_fatal_exception+0x34
/var/tmp/apache2/bin/httpd:0x2a09c
/lib/libc.so.1:0xbfec8
/lib/libc.so.1:0xb4ff4
/lib/libc.so.1:_so_accept+0x8 [ Signal 11 (SEGV)]
/var/tmp/apache2/bin/httpd:unixd_accept+0x10
/var/tmp/apache2/bin/httpd:0x1c2ac
/var/tmp/apache2/bin/httpd:0x1c574
/var/tmp/apache2/bin/httpd:0x1c644
/var/tmp/apache2/bin/httpd:ap_mpm_run+0x76c
/var/tmp/apache2/bin/httpd:main+0x63c
/var/tmp/apache2/bin/httpd:_start+0x5c
[Sat Feb  4 20:36:05 2006] pid 23514 mod_backtrace end of backtrace
```

GDB macros

- Apache comes with numerous GDB macros to print brigades, buckets, strings, filters, memnodes, tables, and process and server records
- Macros are located in `$SRCROOT/.gdbinit`
- Can be sourced using the gdb “source” command

Using GDB macros

```
$ httpd -X
```

```
$ gdb -q /usr/apache2/bin/httpd
```

```
(gdb) source apachemacros
```

```
(gdb) show user
```

```
User command dump_bucket:
```

```
dump_bucket_ex $arg0 0
```

```
...
```

```
(gdb) info function ap_pass_brigade
```

```
All functions matching regular expression "ap_pass_brigade":
```

```
File util_filter.c:
```

```
apr_status_t ap_pass_brigade(ap_filter_t *, apr_bucket_brigade *);
```

```
(gdb) break ap_pass_brigade
```


Using GDB macros (cont.)

(gdb) **attach 975**

(gdb) **continue**

(gdb) **backtrace 4**

```
#0 ap_pass_brigade (next=0x129d18, bb=0x139168) at util_filter.c:489
#1 0x000291d4 in ap_http_header_filter (f=0x138568, b=0x139168) at http_protocol.c:1766
#2 0x0003ad5c in ap_pass_brigade (next=0x138568, bb=0x139168) at util_filter.c:512
#3 0x0003d444 in ap_content_length_filter (f=0x138550, b=0x139168) at protocol.c:1248
```

(gdb) **next**

(gdb) **dump_brigade bb**

dump of brigade 0x139168

	type	(address)	length	data addr	contents	rc
0	FILE	(0x0012d918)	2326	0x0012da58	["**unprintable**"]	1
1	EOS	(0x0012daa8)	0	0x00000000		n/a

end of brigade

(gdb) **detach**

Debugging transient problems

What is transient debugging?

- Transient debugging is the art of correlating unacceptable behaviors to specific application and system components
- Several utilities can help with debugging transient problems:
 - Chaosreader
 - Curl
 - Dtrace
 - Ethereal
 - Firefox HTTP Live Headers

Curl

- Versatile command line utility that can be used to debug web-based problems
- Curl contains several advanced options to print protocol headers and connection errors
- Invaluable utility for locating misbehaving servers and applications

Curl example

```
$ curl -v --user-agent "CURL DEBUG (`date`)" -H "X-foo: yikes"  
http://daemons.net
```

```
About to connect() to daemons.net port 80
```

```
Trying 66.148.84.65... * connected
```

```
Connected to daemons.net (66.148.84.65) port 80
```

```
> GET / HTTP/1.1
```

```
User-Agent: CURL DEBUG (Sat Feb  4 23:02:36 EST 2006)
```

```
Host: daemons.net
```

```
Pragma: no-cache
```

```
Accept: */*
```

```
X-foo: yikes
```

```
< HTTP/1.1 200 OK
```

```
< Date: Sun, 05 Feb 2006 04:04:13 GMT
```

```
< Server: Apache
```

```
< Last-Modified: Sun, 20 Jun 2004 14:39:21 GMT
```

```
< ETag: "5c186-912-c108d840"
```

```
< Accept-Ranges: bytes
```

```
< Content-Length: 2322
```

```
< Content-Type: text/html
```

```
...
```

DTrace

- Dynamic tracing facility introduced in Solaris 10
- Can dynamically instrument applications and the Solaris kernel down to the instruction level
- Utilizes 30k+ probes distributed throughout the Solaris kernel
- Designed to be used on production systems
- No overhead when probes aren't enabled

Dtrace script organization

- Dtrace scripts contain one or more probes, an optional predicate, and an optional action to perform (the default action is `trace()`):

```
provider:module:function:name  
/ predicate /  
{  
    action();  
}
```

Dtrace example #1

- Viewing system calls by Apache process

```
$ dtrace -n 'syscall:::entry  
/execname == "httpd"/  
{  
    @calls[probefunc] = count();  
}
```


Dtrace example #2

- Determining WHO called writev

```
$ dtrace -n 'syscall::writev:entry  
/ execname == "httpd" /  
{  
    ustack();  
}'
```

Dtrace example #3

- Tracing execution flow per request

```
#pragma D option flowindent
pid$target::ap_process_request:entry
{
    self->trace = 1;
}
pid$target::ap_process_request:return
{
    self->trace = 0;
}
pid$target:::entry,
pid$target:::return
/ self->trace /
{ }
```

Dtrace example #4

- Tracing execution flow into the kernel

```
#pragma D option flowindent
pid$target::ap_read_request:entry
{
    self->trace = 1;
}
pid$target::ap_read_request:return
{
    self->trace = 0;
}
pid$target:::entry,
pid$target:::return,
fbt:::entry,
fbt:::return
/ self->trace /
{ }
```

Dtrace example #5

- Watching Logical Apache I/O operations

```
syscall::write:entry
/ execname == "httpd" /
{
    printf("Apache wrote (%s) to fd %d (%s\n", probefunc, arg0,
    fds[arg0].fi_pathname);
}
```

```
syscall::read:entry
/ execname == "httpd" /
{
    printf("Apache read (%s) from fd %d (%s)\n", probefunc, arg0,
    fds[arg0].fi_pathname);
}
```

Dtrace example #6

- Measuring write size

```
$ dtrace -n 'syscall::read:return  
/ execname == "httpd" && errno == 0 /  
{  
    @distribution["Average read size"] = quantize(arg1);  
}
```

Dtrace example #7

- Measuring request processing time

```
pid$target::ap_read_request:entry
{
    self->ts = timestamp;
}
pid$target::ap_read_request:return
{
    self->method = arg1 == 0 ? "Unknown" : copyinstr(*(uintptr_t *)copyin(arg1 +
68,4));
    self->uuri = arg1 == 0 ? "Unknown" : copyinstr(*(uintptr_t *)copyin(arg1 +
200,4));
}
pid$target::ap_process_request:return
{
    printf("Processed %s %s in %d microseconds\n", self->method, self->uuri,
        (timestamp - self->ts) / 1000000);
    self->uuri = 0; self->ts = 0;
```

Conclusion

- Debugging is cool!
- Debugging is great!
- Now it's time for me to escape! :-)

Questions?

References

- Apache: <http://apache.org>
- Chaosreader:
<http://users.tpg.com.au/bdgcvb/chaosreader.html>
- Curl: <http://curl.haxx.se/>
- Dtrace: <http://opensolaris.org/os/community/dtrace/>
- Debugging Web Applications:
<http://www.samag.com/articles/2006/0603/>
- Ethereal: <http://www.ethereal.com/>
- mod_backtrace:
http://people.apache.org/~trawick/exception_hook.htm
- Observing I/O Behavior with the DTraceToolkit:
<http://www.samag.com/documents/s=9915/sam0512a/0512a.htm>