Apache Processing and Threading

Tuesday, April 08, 2014 12:02

1. **WSGI Process/Thread Flags**
   Although Apache can make use of a combination of processes and/or threads to handle requests, this is not unique to the Apache web server and the WSGI specification acknowledges this fact. This acknowledgement is in the form of specific key/value pairs which must be supplied as part of the WSGI environment to a WSGI application. The purpose of these key/value pairs is to indicate whether the underlying web server does or does not make use of multiple processes and/or multiple threads to handle requests.

   These key/value pairs are defined as follows in the WSGI specification.
   wsgi.multithread
   This value should evaluate true if the application object may be simultaneously invoked by another thread in the same process, and should evaluate false otherwise.
   wsgi.multiprocess
   This value should evaluate true if an equivalent application object may be simultaneously invoked by another process, and should evaluate false otherwise.

2. **Multi-Processing Modules**
   The main factor which determines how Apache operates is which multi-processing module (MPM) is built into Apache at compile time. Although runtime configuration can customise the behaviour of the MPM, the choice of MPM will dictate whether or not multithreading is available.

   On UNIX based systems, Apache defaults to being built with the 'prefork' MPM. If Apache 1.3 is being used this is actually the only choice, but for later versions of Apache, this can be overridden at build time by supplying an appropriate value in conjunction with the '–with-mpm' option when running the 'configure' script for Apache. The main alternative to the 'prefork' MPM which can be used on UNIX systems is the 'worker' MPM.

   If you are unsure which MPM is built into Apache, it can be determined by running the Apache web server executable with the '-V' option. The output from running the web server executable with this option will be information about how it was configured when built:
```
Server version: Apache/2.2.1
Server built:   Mar  4 2007 20:48:15
Server's Module Magic Number: 20051115:1
Server loaded:  APR 1.2.6, APR-Util 1.2.6
Compiled using: APR 1.2.6, APR-Util 1.2.6
Architecture:   32-bit
Server MPM:     Worker
  threaded:     yes (fixed thread count)
    forked:     yes (variable process count)
Server compiled with....
 -D APACHE_MPM_DIR="server/mpm/worker"
 -D APR_HAS_MMAP
 -D APR_HAVE_IPV6 (IPv4-mapped addresses enabled)
 -D APR_USE_SYSVSEM_SERIALIZE
 -D APR_USE_PTHREAD_SERIALIZE
 -D SINGLE_LISTEN_UNSERIALIZED_ACCEPT
 -D APR_HAS_OTHER_CHILD
 -D AP_HAVE_RELIABLE_PIPED_LOGS
 -D DYNAMIC_MODULE_LIMIT=128
 -D HTTPD_ROOT="/usr/local/apache-2.2"
 -D SUEXEC_BIN="/usr/local/apache-2.2/bin/suexec"
 -D DEFAULT_SCOREBOARD="logs/apache_runtime_status"
 -D DEFAULT_ERRORLOG="logs/error_log"
 -D AP_TYPES_CONFIG_FILE="conf/mime.types"
 -D SERVER_CONFIG_FILE="conf/httpd.conf"
```

3. **The UNIX 'prefork' MPM**
   This MPM is the most commonly used. It was the only mode of operation available in Apache 1.3 and is still the default mode on UNIX systems in later versions of Apache. In this configuration, the main Apache process will at startup create multiple child processes. When a request is received by the parent process, it will be processed by which ever of the child processes is ready.

   Each child process will only handle one request at a time. If another request arrives at the same time, it will be handled by the next available child process. When it is detected that the number of available processes is running out, additional child processes will be created as necessary. If a limit is specified as to the number of child processes which may be created and the limit is reached, plus there are sufficient requests arriving to fill up the listener socket queue, the client may instead receive an error resulting from not being able to establish a connection with the web server.

   Where additional child processes have to be created due to a peak in the number of current requests arriving and where the number of requests has subsequently dropped off, the excess child processes may be shutdown and killed off. Child processes may also be shutdown and killed off after they have handled some set number of requests.
   - **multithread: False**
   - **multiprocess: True**

4. **The UNIX 'worker' MPM**
   The 'worker' MPM is similar to 'prefork' mode except that within each child process there will exist a number of worker threads. Instead of a request only being able to be processed by the next available idle child process and with the handling of the request being the only thing the child process is then doing, the request may be processed by a worker thread within a child process which already has other worker threads handling other requests at the same time.

   It is possible that a WSGI application could be executed at the same time from multiple worker threads within the one child process. This means that multiple worker threads may want to access common shared data at the same time. As a consequence, such common shared data must be protected in a way that will allow access and modification in a thread safe manner. Normally this would necessitate the use of some form of synchronisation mechanism to ensure that only one thread at a time accesses and or modifies the common shared data.

   If all worker threads within a child process were busy when a new request arrives the request would be processed by an idle worker thread in another child process. Apache may still create new child processes on demand if necessary. Apache may also still shutdown and kill off excess child processes, or child processes that have handled more than a set number of requests.

   Overall, use of 'worker' MPM will result in less child processes needing to be created, but resource usage of individual child processes will be greater. On modern computer systems, the 'worker' MPM would in general be the prefered MPM to use and should if possible be used in preference to the 'prefork' MPM.

   Although contention for the global interpreter lock (GIL) in Python can causes issues for pure Python programs, it is not generally as big an issue when using Python within Apache. This is because all the underlying infrastructure for accepting requests and mapping the URL to a WSGI application, as well as the handling of requests against static files are all performed by Apache in C code. While this code is being executed the thread will not be holding the Python GIL, thus allowing a greater level of overlapping execution where a system has multiple CPUs or CPUs with multiple cores.

   This ability to make good use of more than processor, even when using multithreading, is further enchanced by the fact that Apache uses multiple processes for handling requests and not just a single process. Thus, even when there is some contention for the GIL within a specific process, it doesn't stop other processes from being able to run as the GIL is only local to a process and does not extend across processes.

- **multithread: True**
- **multiprocess: True**

5. **The Windows 'winnt' MPM**
   On the Windows platform the 'winnt' MPM is the only option available. With this MPM, multiple worker threads within a child process are used to handle all requests. The 'winnt' MPM is different to the 'worker' mode however in that there is only one child process. At no time are additional child processes created, or that one child process shutdown and killed off, except where Apache as a whole is being stopped or restarted. Because there is only one child process, the maximum number of threads used is much greater.

   The WSGI environment key/value pairs indicating how processes and threads are being used will for this configuration be as follows.
   - **multithread: True**
   - **multiprocess: False**