ModuleLife

HTTPd start-up phase:

- First execution "configuration check"
  - The module is dynamically loaded in the single-process single-thread server, that is still running as root if it was started as root.
  - The configuration is parsed, but just to check that it is valid (sanity check).
  - The module post_init hook is called for the 1st time in the sanity check.
  - Depending on platform, the module is then completely unloaded. It means that global and static variables are reset. This also means any cleanups left pointing into the module against the process pool will go boom on exit.
- Second execution "live"
  - The module is loaded again if necessary.
  - The configuration is parsed again, for real this time.
  - The module post_init hook is called for the 2nd time.
  - Children processes are started. They may be forked (Unix) or spawned (Windows).
  - The pre_init/post_init hooks are never called in the forked (Unix) child process, see the child_init hook instead.
  - If the children are spawned (as on Windows), they repeat the whole startup, and the post_config hook is called exactly twice again in the children (and never a third time on graceful restart).

HTTPd restart:

- On graceful restart, the module is unloaded and reloaded/init, and the post_config hook is called (still in the parent process) once, not twice.
- On Windows, a new spawned child process is created by the parent upon a graceful restart.
- During a graceful restart, new children processes may be running on the new configuration while old children on the old configuration are still serving out their requests.

1st and next post_config hook calls

A common hack to execute the post_config hook body only once is to set a flag on 1st execution and exit. On 2nd execution, the flag is retrieved and module initialization is performed. This hack uses the apr_pool_userdata_set and apr_pool_userdata_get functions to saves the flag in memory managed by the process pool.

- Example from http://www.codemass.com/mod_shm_counter/
- See also http://www.clove.org/~aaron/presentations/apachecon2002/

```
static int shm_counter_post_config(apr_pool_t *pconf, apr_pool_t *plog,
                                   apr_pool_t *ptemp, server_rec *s)
{
    apr_status_t rv;
    shm_counter_scfg_t *scfg;
    void *data = NULL;
    const char *userdata_key = "shm_counter_post_config";

    /* Apache loads DSO modules twice. We want to wait until the second
     * load before setting up our global mutex and shared memory segment.
     * To avoid the first call to the post_config hook, we set some
     * dummy userdata in a pool that lives longer than the first DSO
     * load, and only run if that data is set on subsequent calls to
     * this hook. */
    apr_pool_userdata_get(&data, userdata_key, s->process->pool);
    if (data == NULL) {
        /* WARNING: This must *not* be apr_pool_userdata_setn(). The
         * reason for this is because the static symbol section of the
         * DSO may not be at the same address offset when it is reloaded.
         * Since setn() does not make a copy and only compares addresses,
         * the get() will be unable to find the original userdata. */
        apr_pool_userdata_set((const void *)1, userdata_key,
                              apr_pool_cleanup_null, s->process->pool);
        return OK; /* This would be the first time through */
    }

    /* If we made it this far, we can safely initialize the module */
```

A problem with this hack exists when a module is installed while the server is running, and the server is just restarted to load the module. In this case, the post_config hook is executed once after restart and will only set the flag. Module initialization will not be performed, and when child_init and other hooks are called, bad things may happen.

post_config hook call by children processes

On WinNT MPM, the post_config hook is also called by the children processes.

To know if we are in a child, we can check the AP_PARENT_PID environment variable: it is only set in children on the WinNT MPM.

Note: Some module take for granted that the post_config hook is only called in the parent process. That's wrong (on Win32). But calling apr_global_mutex_create() in the parent and in the child with the same filename does not hurt, the mutex is shared.

- Example from http://www.codemass.com/mod_shm_counter/

```
/* If we made it this far, we can safely initialize the module */
scfg = ap_get_module_config(s->module_config, &shm_counter_module);

ap_add_version_component(pconf, "mod_shm_counter/$Revision $");

/* Since we are still in the parent before any children have been
 * created, it is safe to create the shared lock and shared segment.
 * If we waited until a child had been created, we run in to a race
 * condition.
 */

rv = apr_global_mutex_create(&scfg->mutex, scfg->shmcounterlockfile,
                             APR_LOCK_DEFAULT, pconf);
```

Race conditions during graceful restart

During a graceful restart, old children are still serving old requests while new children are serving new requests. If the same lock must be used by old and new children, then the lock name must be the same and cannot be generated with tmpnam() or similar functions in the post_config hook.

Preventing module unloading

To prevent module unloading, there is a *sick and twisted hack* (wrowe dixit): "the evil method is to use apr_dso_load() against the process pool to load it 'permanently'".