RPC XID issues

Ric Werme
werme@zk3.dec.com

Digital Equipment Corp.

RPC XID use and misuse

- **XID, that’s, uh, something that takes care of itself?**
- **It’s not NFS, talk abut something interesting!**
- **Something that never repeats.**

The Duplicate Request Cache

- **Remembers replies sent for non-idempotent calls.**
- **Cache based on not only XID, but also:**
  - **Client's IP address**
  - **RPC program, version, and procedure number**
- **Size (in Digital UNIX): 512 to 4096 entries**

The original XID formulae

- **libc (clntudp_bufcreate):**
  - `call_msg.rm_xid = getpid() ^ now.tv_sec ^ now.tv_usec`
  - `(*(u_short *)(cu->cu_outbuf))++;`
  - tv_usec may be a small subset of 1,000,000.
  - Can jump backward every second!
- **kernel:**
  - `clntxid = time.tv_sec ^ time.tv_usec`
  - `#define alloc_xid() (clntxid++)`

XIDs vs. user level NFS

- 20 bit initial selection (size of tv_usec)
- Big endian - increments low two bytes
- Little endian - increments high two bytes
- 64 K number space rut

NFS from OS/2

- **Increments a short (ffff887f -> 0000887f)**
- **Other two bytes hold only a few values**
  - **887f, 376c, 378e**

How we handle XIDs today

- **Include client's port number in DRC**
- **120 second lifetime (I thought I used TCP's 2*MSL)**
- **Smarter XID generation in kernel**

```
clntxid = time.tv_sec << 12;
```

- **Smarter XID generation in libc**

```
_clnt_xid = (((u_int) now.tv_sec     // sec
              * 100)    // * ticks/(hour/600)
              /6)       // * hour*600/sec
             % 60000;   // scale to fit
*(u_int *)(cu->cu_outbuf) =
    ++_clnt_xid + ((u_int) getpid() << 16);
```

Libc detail

- **Two types of clients**
- **Fast, furious, and shortlived**
  - **ypcat**
  - **pid can be reused**
  - **Need block of XIDs available to client**
- **Slow, relaxed, and longlived**
  - **lockd, statd**
  - **pid unlikely to be reused**
  - **Need something that doesn’t repeat over time**

Three easy pieces

- **Give each pid a 64K number space**

  ```
  getpid() << 16
  ```

- **Discourage reuse for an hour**

  seconds * 60000 / 3600

  ```
  (((u_int) now.tv_sec    // sec
    * 100)  // * ticks/(hour/600)
   /6)      // * hour*600/sec
  % 60000;  // scale to fit
  ```

- **remainder for fast, short processes**

  1 second = 60000 / 3600 ticks

  1 second ~= 17 ticks