15-440 Recitation/Lecture Thing.

Plans

- Today: recitation style going over the second lab
- Find a partner, email staff-440 with your andrew ids
- Tomorrow: lecture with Kesden during normal recitation time

Summary

- There are 6 parts to this lab.
- The first part is due next Wednesday.
- There is a large code base, START ASAP.
- The last 5 parts are due March 24.

You get to build a DFS!

- 6 stages that build on each other:
  - Lock server, at-most-once RPC semantics
  - Extent server, create/lookup/readdir
  - read/write/open/setattr
  - mkdir/unlink, more lock stuff
  - caching locks
  - caching extents

Today: Part 1

- Implementing the lock server
  - provide mutual exclusion
  - use pthread mutexes and condition variables

- Implementing at-most-once RPC semantics

Vms! Yay!

- Nifty!
- VirtualBox
- We provide the image for you
- If you want other packages, the password is 'systems'
- sudo apt-get install package
- demo...

Dist Mutual Exclusion

- Lock is a 64-bit number
  - the client would:

    acquire(lock_a) ; do work; release(lock_a);

- Create the lock if it doesn't exist on the server

What we give you

- Simple RPC framework, a skeleton for the lock server
- Sets up sockets, marshalling/unmarshalling, and sending stuff over TCP
- BUT it does not keep track of the RPC request state, so duplicate RPCs are invoked twice!

Example RPC call

```
lock_protocol::status
lock_server::stat(int clt, lock_protocol::lockid_t lid,
int &r)
{
    lock_protocol::status ret = lock_protocol::OK;
    printf("stat request from clt %d\n", clt);
    r = nacquire;
    return ret;
}
```

Your Job

```
lock_protocol::status
lock_server::acquire(int clt, lock_protocol::lockid_t
lid, int &r)
{}

lock_protocol::status
lock_server::release(int clt, lock_protocol::lockid_t
lid, int &r)
{}
```

Figuring out if it works

- RPC_LOSSY: drops, duplicates, delays
- Run lock_tester with RPC_LOSSY=0

- Run lock_tester with RPC_LOSSY=5

  Should fail!

Why does it fail?

- At-most-once RPC semantics have not yet been implemented
- If the reply was dropped, a duplicate is sent
  - acquire(a); acquire(a)

Implementing at-most-once RPC

- Start the timeout thread in the rpc server constructor
- On the server side, manage the state of the RPCs sent and the replies

Naïve Approach

- Remember every RPC call (the client sends a unique RPC identifier)
- Remember every RPC reply (to avoid invoking the actual function)
- What's the problemo?

Sliding Window RPCs

Client sends:

```
marshall m1 << clt nonce //client id
            << srv nonce //server id
            << proc //procedure # (acquire,etc)
            << myxid //unique request id for this RPC
            << xid_rep_window.front() //Last out of order
                RPC reply received
            << req.str() //Data
```

How to use this info

- You have to make sure of the client id, the xid, and the last out of order RPC
  - check whether the request is new, done, in progress, or forgotten
  - figure out which replies you can forget
  - keep track of replies of local RPC calls to ensure at-most-once semantics

Other Stuff

- You'll be required to know a bit about C++ STL data structures for this project.
  - not to fear, the internet is here
- pthread mutexes, condition variables
  - read the man pages!
  - make sure you initialize them

You should start ASAP!

Go home and read the handouts!

Email us with your partner!

SVN review

- svn add
- svn commit
- svn update
- svn copy

We're here to help!

Email us with questions!