6.033 Spring 2018

Lecture #15

- **When replication fails us**
  - **Atomicity via shadow copies**
  - **Isolation**
  - **Transactions**

high-level goal: build reliable systems from unreliable components

this is difficult because reasoning about failures is difficult. we need some abstractions that will let us simplify.

atomicity

an action is atomic if it **happens completely or not at all**. if we can guarantee atomicity, it will be much easier to reason about failures

```
transfer (bank, account_a, account_b, amount):
    bank[account_a] = bank[account_a] - amount
    bank[account_b] = bank[account_b] + amount
```

← **crash!** 💥

problem: `account_a` lost **`amount`** dollars, but `account_b` didn't gain **`amount`** dollars

```
transfer (bank, account_a, account_b, amount):
    bank[account_a] = bank[account_a] - amount
    bank[account_b] = bank[account_b] + amount
```

← **crash!** 💥

solution: make this action atomic. ensure that we complete both steps or neither step.

quest for atomicity: attempt 1

```
transfer (bank_file, account_a, account_b, amount):
    bank = read_accounts(bank_file)
    bank[account_a] = bank[account_a] - amount
    bank[account_b] = bank[account_b] + amount
    write_accounts(bank_file)
```

crash! 💥

quest for atomicity: attempt 1

```
transfer (bank_file, account_a, account_b, amount):
    bank = read_accounts(bank_file)
    bank[account_a] = bank[account_a] - amount
    bank[account_b] = bank[account_b] + amount
    write_accounts(bank_file) ← crash! 💥
```

problem: a crash during `write_accounts` leaves `bank_file` in an intermediate state

quest for atomicity: attempt 2

(shadow copies)

```
transfer (bank_file, account_a, account_b, amount):
    bank = read_accounts(bank_file)
    bank[account_a] = bank[account_a] - amount
    bank[account_b] = bank[account_b] + amount
    write_accounts(tmp_file)
    rename(tmp_file, bank_file)
```

crash! 💥

quest for atomicity: attempt 2

(shadow copies)

```
transfer (bank_file, account_a, account_b, amount):
    bank = read_accounts(bank_file)
    bank[account_a] = bank[account_a] - amount
    bank[account_b] = bank[account_b] + amount
    write_accounts(tmp_file) ← crash! 💥
    rename(tmp_file, bank_file)
```

quest for atomicity: attempt 2

(shadow copies)

```
transfer (bank_file, account_a, account_b, amount):
    bank = read_accounts(bank_file)
    bank[account_a] = bank[account_a] - amount
    bank[account_b] = bank[account_b] + amount
    write_accounts(tmp_file)
    rename(tmp_file, bank_file) ← crash! 💥
```

problem: a crash during `rename` potentially leaves `bank_file` in an intermediate state

quest for atomicity: attempt 2

(shadow copies)

```
transfer (bank_file, account_a, account_b, amount):
    bank = read_accounts(bank_file)
    bank[account_a] = bank[account_a] - amount
    bank[account_b] = bank[account_b] + amount
    write_accounts(tmp_file)
    rename(tmp_file, bank_file)  ← crash! 💥
```

solution: make `rename` atomic

quest for atomicity: making rename atomic

```
directory entries
   filename “bank_file” -> inode 1
   filename “tmp_file” -> inode 2

inode 1: // old data
    data blocks: [..]
    refcount: 1

inode 2: // new data
    data blocks: [..]
    refcount: 1
```

```
rename(tmp_file, orig_file):
    // point bank_file’s dirent at inode 2
    // delete tmp_file’s dirent
    // remove refcount on inode 1
```

quest for atomicity: making `rename` atomic

```
directory entries
   filename "bank_file" -> inode 1
   filename "tmp_file" -> inode 2

inode 1: // old data              inode 2: // new data
    data blocks: [..]                 data blocks: [..]
    refcount: 1                       refcount: 1
```

```
rename(tmp_file, orig_file):
    tmp_inode = lookup(tmp_file)   // = 2
    orig_inode = lookup(orig_file) // = 1

    // point bank_file's dirent at inode 2
    // delete tmp_file's dirent
    // remove refcount on inode 1
```

quest for atomicity: making `rename` atomic

```
directory entries
    filename "bank_file" -> inode 2
    filename "tmp_file" -> inode 2

inode 1: // old data                    inode 2: // new data
    data blocks: [..]                       data blocks: [..]
    refcount: 1                             refcount: 1
```

```
rename(tmp_file, orig_file):
    tmp_inode = lookup(tmp_file)   // = 2
    orig_inode = lookup(orig_file) // = 1

    orig_file dirent = tmp_inode
    // delete tmp_file's dirent
    // remove refcount on inode 1
```

quest for atomicity: making `rename` atomic

```
directory entries
    filename "bank_file" -> inode 2
```

```
inode 1: // old data
    data blocks: [..]
    refcount: 1
```

```
inode 2: // new data
    data blocks: [..]
    refcount: 1
```

```
rename(tmp_file, orig_file):
    tmp_inode = lookup(tmp_file)   // = 2
    orig_inode = lookup(orig_file) // = 1

    orig_file dirent = tmp_inode
    remove tmp_file dirent
    // remove refcount on inode 1
```

quest for atomicity: making `rename` atomic

```
directory entries
    filename "bank_file" -> inode 2
```

```
inode 1: // old data
    data blocks: [..]
    refcount: 0
```

```
inode 2: // new data
    data blocks: [..]
    refcount: 1
```

```
rename(tmp_file, orig_file):
    tmp_inode = lookup(tmp_file)   // = 2
    orig_inode = lookup(orig_file) // = 1

    orig_file dirent = tmp_inode
    remove tmp_file dirent
    decref(orig_inode)
```

quest for atomicity: making `rename` atomic

```
directory entries
    filename "bank_file" -> inode 1
    filename "tmp_file" -> inode 2

inode 1: // old data                inode 2: // new data
    data blocks: [..]                   data blocks: [..]
    refcount: 1                         refcount: 1
```

```
rename(tmp_file, orig_file):
    tmp_inode = lookup(tmp_file)    // = 2
    orig_inode = lookup(orig_file) // = 1
                                        <-- crash! 💥
    orig_file dirent = tmp_inode
    remove tmp_file dirent
    decref(orig_inode)
```

rename didn't happen

quest for atomicity: making `rename` atomic

```
directory entries
    filename "bank_file" -> inode 2
    filename "tmp_file" -> inode 2

inode 1: // old data                inode 2: // new data
    data blocks: [..]                   data blocks: [..]
    refcount: 1                         refcount: 1
```

```
rename(tmp_file, orig_file):
    tmp_inode = lookup(tmp_file)   // = 2
    orig_inode = lookup(orig_file) // = 1

    orig_file dirent = tmp_inode
    remove tmp_file dirent
    decref(orig_inode)
```

crash! 💥

rename happened,
but refcounts are wrong

quest for atomicity: making `rename` atomic

```
directory entries
    filename "bank_file" -> inode ?
    filename "tmp_file" -> inode 2

inode 1: // old data              inode 2: // new data
    data blocks: [..]                 data blocks: [..]
    refcount: 1                       refcount: 1
```

```
rename(tmp_file, orig_file):
    tmp_inode = lookup(tmp_file)    // = 2
    orig_inode = lookup(orig_file) // = 1

    orig_file dirent = tmp_inode
    remove tmp_file dirent
    decref(orig_inode)
```

← **crash!** 💥

crash *during* this line seems bad.. but is okay because single-sector writes are themselves atomic

quest for atomicity: making `rename` atomic

```
directory entries
    filename "bank_file" -> inode 2
    filename "tmp_file" -> inode 2

inode 1: // old data                inode 2: // new data
    data blocks: [..]                   data blocks: [..]
    refcount: 1                         refcount: 1
```

```
rename(tmp_file, orig_file):
    tmp_inode = lookup(tmp_file)    // = 2
    orig_inode = lookup(orig_file) // = 1

    orig_file dirent = tmp_inode
    remove tmp_file dirent
    decref(orig_inode)
```

crash! 💥

rename happened,
but refcounts are wrong

quest for atomicity: making `rename` atomic

```
directory entries
   filename "bank_file" -> inode 2
   filename "tmp_file" -> inode 2

inode 1: // old data              inode 2: // new data
    data blocks: [..]                 data blocks: [..]
    refcount: 1                       refcount: 1
```

```
rename(tmp_file, orig_file):
    tmp_inode = lookup(tmp_file)    // = 2
    orig_inode = lookup(orig_file) // = 1
    incref(tmp_inode)
    orig_file dirent = tmp_inode
    decref(orig_inode)
    remove tmp_file dirent
    decref(tmp_inode)
```

problem: this is a mess, and is still incorrect

solution: recover from failure

(clean things up)

```
recover(disk):
    for inode in disk.inodes:
        inode.refcount = find_all_refs(disk.root_dir, inode)
    if exists("tmp_file"):
        unlink("tmp_file")
```

atomicity

(first abstraction)

not quite solved; shadow copies perform poorly even for a single user and a single file, and we haven’t even talked about concurrency

isolation

(second abstraction)

if we guarantee isolation, then two actions A1 and A2 will appear to have run **serially** even if they were executed concurrently
(i.e., A1 before A2, or vice versa)

transactions: provide atomicity and isolation

```
Transaction 1
  begin
  transfer(A, B, 20)
  withdraw(B, 10)
  end
```

```
Transaction 2
  begin
  transfer(B, C, 5)
  deposit(A, 5)
  end
```

atomicity: each transaction will appear to have run to completion, or not at all

isolation: when multiple transactions are run concurrently, it will appear as if they were run sequentially (serially)

atomicity and isolation — and thus, transactions — make it easier to reason about failures (and concurrency)

```
transfer (bank_file, account_a, account_b, amount):
    acquire(lock)
    bank = read_accounts(bank_file)
    bank[account_a] = bank[account_a] - amount
    bank[account_b] = bank[account_b] + amount
    write_accounts("tmp_file")
    rename("tmp_file", bank_file)
    release(lock)
```

couldn't we just put locks around everything?

(isn't that what locks are *for*?)

```
transfer (bank_file, account_a, account_b, amount):
    acquire(lock)
    bank = read_accounts(bank_file)
    bank[account_a] = bank[account_a] - amount
    bank[account_b] = bank[account_b] + amount
    write_accounts("tmp_file")
    rename("tmp_file", bank_file)
    release(lock)
```

this particular strategy will perform poorly

(would force a single transfer at a time)

locks sometimes require global reasoning, which is messy

eventually, we'll incorporate locks, but in a systematic way

goal: to implement **transactions**, which provide atomicity and isolation, while not hindering performance

atomicity → **shadow copies.** work, but perform poorly and don't allow for concurrency

isolation → **?** (coarse-grained locks perform poorly, finer-grained locks are difficult to reason about)

eventually, we also want transaction-based systems to be **distributed**: to run across multiple machines

- **Transactions** provide **atomicity** and **isolation**, both of which make it easier for us to reason about failures because we don't have to deal with intermediate states.

- **Shadow copies** are one way to achieve atomicity. The work, but perform poorly: require copying an entire file even for small changes, and don't allow for concurrency.