In-class: xv6 log

This assignment explores the xv6 log in two parts. First, you'll artificially create a crash which illustrates why logging is needed. Second, you'll remove one inefficiency in the xv6 logging system.

Creating a Problem

The point of the xv6 log is to cause all the disk updates of a filesystem operation to be atomic with respect to crashes. For example, file creation involves both adding a new entry to a directory and marking the new file's inode as in-use. A crash that happened after one but before the other would leave the file system in an incorrect state after a reboot, if there were no log.

The following steps will break the logging code in a way that leaves a file partially created.

First, replace `commit_trans()` in `log.c` with this code:

```
#include "mmu.h"
#include "proc.h"
void
commit_trans(void)
{
  if (log.lh.n > 0) {
    write_head();
    if(proc->pid > 1)        // AAA
      log.lh.sector[0] = 0; // BBB
    install_trans();
    if(proc->pid > 1)        // AAA
      panic("commit_trans mimicking crash"); // CCC
    log.lh.n = 0;
    write_head();
  }

  acquire(&log.lock);
  log.busy = 0;
  wakeup(&log);
  release(&log.lock);
}
```

The BBB line causes the first sector in the log to be written to sector zero, rather than wherever it should be written. During file creation, the first sector in the log is the new file's inode updated to have non-zero `type`. Line BBB causes the block with the updated inode to be written to sector 0 (whence it will never be read), leaving the on-disk inode still marked unallocated. The CCC line forces a crash. The AAA lines suppress this buggy behavior for `init`, which creates files before the shell starts.

Second, replace `recover_from_log()` in `log.c` with this code:

```
static void
recover_from_log(void)
{
  read_head();
  cprintf("recovery: n=%d but ignoring\n", log.lh.n);
  // install_trans();
  log.lh.n = 0;
  // write_head();
}
```

This modification suppresses the log recovery (which would repair the damage caused by your change to `commit_trans()`).

Finally, remove the `-snapshot` option from the definition of `QEMUOPTS` in your Makefile so that the disk image will see the changes:

```
QEMUOPTS = -hdb fs.img xv6.img -smp $(CPUS) -m 512 $(QEMUEXTRA)
```

Now remove `fs.img` and run xv6:

```
% rm fs.img ; make qemu
```

Tell the xv6 shell to create a file:

```
$ echo hi > a
```

You should see the panic from `commit_trans()`. So far it is as if a crash occurred in a non-logging system in the middle of creating a file.

Now re-start xv6, keeping the same `fs.img`:

```
% make qemu
```

And look at file `a`:

```
$ cat a
```

Something should go wrong. Make sure you understand what happened. Which of the file creation's modifications were written to the disk before the crash, and which were not?

Solving the Problem

Now fix `recover_from_log()`:

```
static void
recover_from_log(void)
{
  read_head();
  cprintf("recovery: n=%d\n", log.lh.n);
  install_trans();
  log.lh.n = 0;
  write_head();
}
```

Run xv6 (keeping the same `fs.img`) and read `a` again:

```
$ cat a
```

This time there should be no crash. Make sure you understand why the file system now works.

Why was the file empty, even though you created it with `echo hi > a`?

Now remove your modifications to `commit_trans()` (the if's and the AAA and BBB lines), so that logging works again, and remove `fs.img`.

Streamlining Commit

Suppose the file system code wants to update an inode in block 3. The file system code will call `bp=bread(sector 3)`, update the buffer data, then call `log_write(bp)`. `log_write()` will copy the data to a block in the log on disk, for example sector 999. When the file system code commits, `install_trans()` reads sector 999 from the log, copies its contents into the in-memory buffer for sector 3, and then writes that buffer to sector 3 on the disk.

However, in `install_trans()`, it turns out that the modified sector 3 is guaranteed to be still in the buffer cache, where the file system code left it. Make sure you understand why it would be a mistake for the buffer cache to evict sector 3 from the buffer cache before the commit.

Since the modified sector 3 is guaranteed to already be in the buffer cache, there's no need for `install_trans()` to read sector 999 from the log. Your job: modify `log.c` so that, when `install_trans()` is called from `commit_trans()`, `install_trans()` does not perform the needless read from the log.

To test your changes, create a file in xv6, restart, and make sure the file is still there.

Challenge problem: `log_write()` writes each log sector to disk immediately, and `commit_trans()` (and `install_trans()`) always write the file system blocks to the disk right away. Figure out how to improve performance by delaying these writes, and modify xv6 accordingly.

Challenge problem: allow multiple concurrent transactions.

Submit: your modified log.c

MIT OpenCourseWare
http://ocw.mit.edu

6.828 Operating System Engineering

Fall 2012

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.