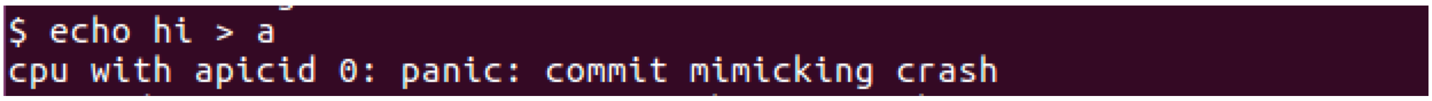
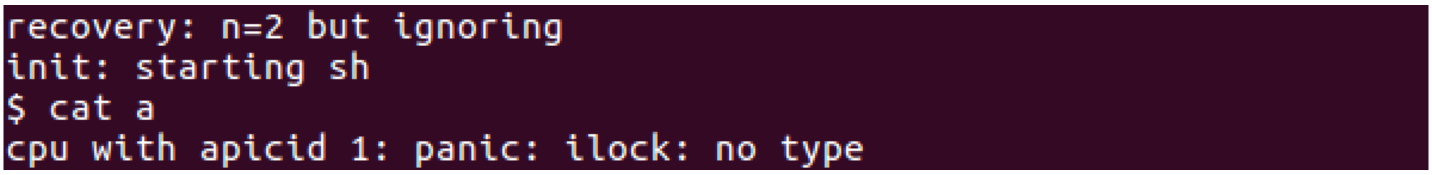
Homework 6 : Xv6 Logging

Breaking the logging code

Output :



```
$ echo hi > a
cpu with apicid 0: panic: commit mimicking crash
```



```
recovery: n=2 but ignoring
init: starting sh
$ cat a
cpu with apicid 1: panic: ilock: no type
```

> What happened?

As said in the assignment, the "BBB" line in the buggy version of `commit()` causes `block[0]` to be written to block `0`, instead of the proper location. Then the crash happens.

Then, since `recover_from_log()` does essentially nothing, we boot up with the disk still in an inconsistent state.

When we execute `cat a`, we will have the following calls : `cat()` → `read()` → `sys_read()` → `fileread()`, which eventually calls `ilock(f->ip)`. And here in `ilock()`, the check `if(ip->type == 0)` will fail, because the first block which holds the type hasn't been written where it was supposed to and thus the data it holds is "not found" here. This results in a call to `panic("ilock: no type");`, which produces the output from above.

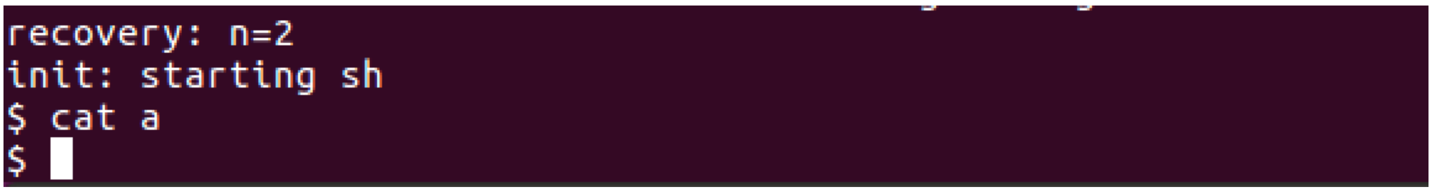
> Which file creation modifications were written to disk and which were not?

Since `commit()` executed `write_log()`, `write_head()` and `install_trans()`, the log content and log header were written to the disk before crashing (ie. reaching `panic()`. In particular, since `log.lh.block[0] = 0` preceedes the `install_trans()` call, its effects were written to disk too.

Since `log.lh.n = 0` and the second `write_head()` come after `panic()`, they were not executed and thus not written to disk.

Fixing logging

Output :



```
recovery: n=2
init: starting sh
$ cat a
$ 
```

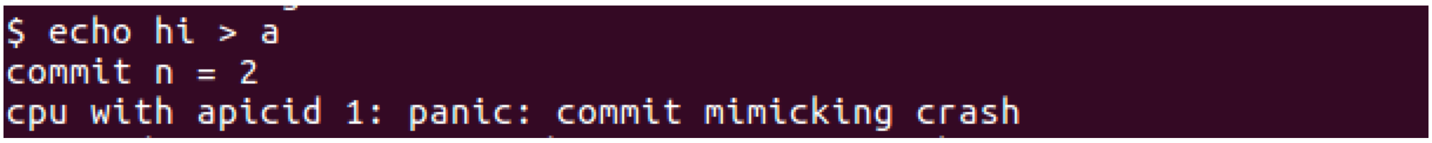
> This time there is no crash. Why?

Notice that in `commit()`, the header are written before we make the buggy modification. Since `recover_from_log()` calls `read_head()` and restores from that, it restores the disk in a consistent state.

> Why was the file empty if you created it with `echo hi > a`?

Because the we `panic` before even having logged modifications to store "hi" (ie. we only created the file `a`). This can be shown this way :

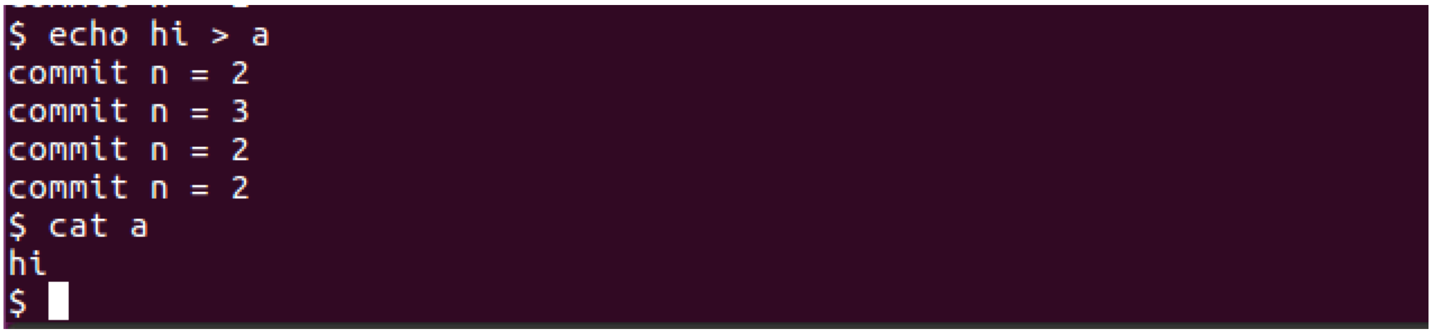
If we add a `cprintf()` at the entry of the `commit()` function, we get the following output :



```
$ echo hi > a
commit n = 2
cpu with apicid 1: panic: commit mimicking crash
```

And an empty file `a`, as before.

Now if we comment out the `panic()` call, we get :



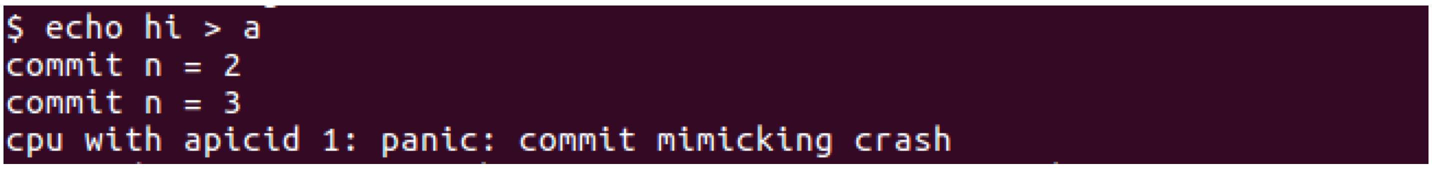
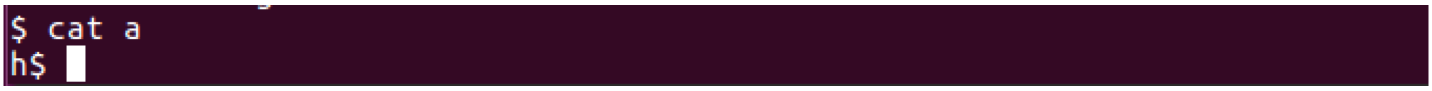
```
$ echo hi > a
commit n = 2
commit n = 3
commit n = 2
commit n = 2
$ cat a
hi
$ 
```

We see that commit is called 4 times, and that `"hi"` is indeed written in file `a`.

One easy way to stop after having logged a few more steps here is to panic when we see `log.lh.n == 3`. The edited code :

```c
void
commit(void)
{
  int pid = proc->pid;
  if (log.lh.n > 0) {
    write_log();
    write_head();
    cprintf("commit n = %d\n", log.lh.n);
    if(pid > 1)            // AAA
      log.lh.block[0] = 0; // BBB
    install_trans();
    if(pid > 1 && log.lh.n == 3)            // AAA
      panic("commit mimicking crash"); // CCC
    log.lh.n = 0;
    write_head();
  }
}
```

Then we get this :


```
$ echo hi > a
commit n = 2
commit n = 3
cpu with apicid 1: panic: commit mimicking crash
```

```
$ cat a
h$ 
```

As we see here, if we panic a bit later, `"h"` is saved in `a`.

Improving commit

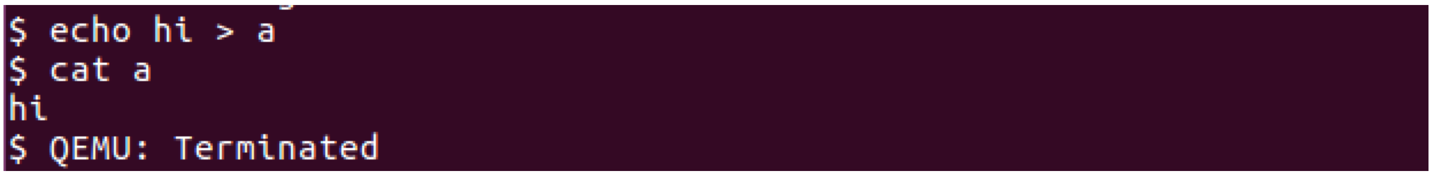
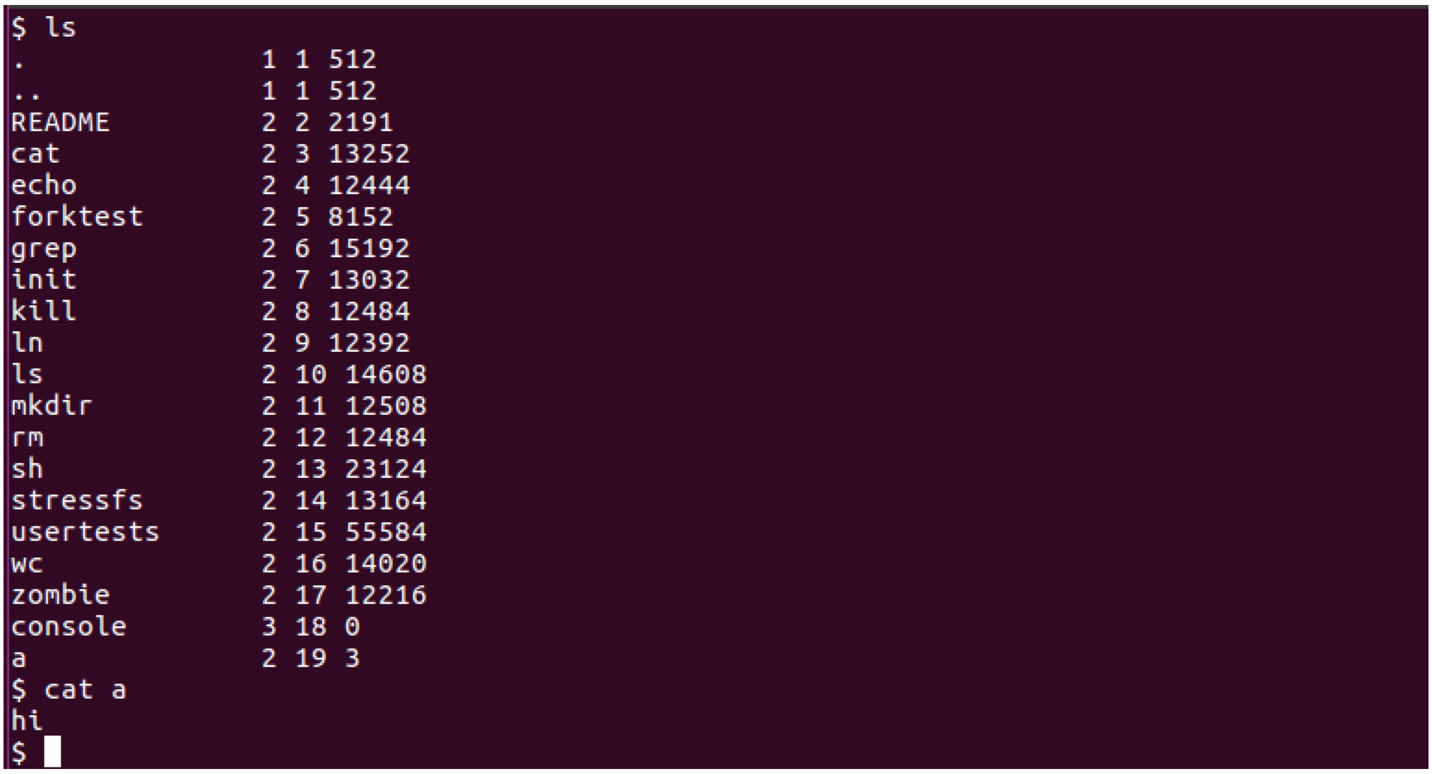
> Why would it be a mistake for the buffer cache to evict block 33 from the buffer cache before the commit?

I couldn't figure this one out...

Modifications :

- Changed `install_trans()` to take a parameter `from_commit`, which tells it whether it was called from `commit()` or not.
- Modified calls to `install_trans()` accordingly in `commit()` and `recover_from_log()`

Test : *As suggested in the assignment*

Create a file :

```
$ echo hi > a
$ cat a
hi
$ QEMU: Terminated
```
Then restart and make sure the file is still there :

```
$ ls
.              1 1 512
..             1 1 512
README         2 2 2191
cat            2 3 13252
echo           2 4 12444
forktest       2 5 8152
grep           2 6 15192
init           2 7 13032
kill           2 8 12484
ln             2 9 12392
ls             2 10 14608
mkdir          2 11 12508
rm             2 12 12484
sh             2 13 23124
stressfs       2 14 13164
usertests      2 15 55584
wc             2 16 14020
zombie         2 17 12216
console        3 18 0
a              2 19 3
$ cat a
hi
$ 
```