

XV6 ASSIGNMENT REPORT

ABHAY SHANKAR K: CS21BTECH11001

1. OVERVIEW

We have made a single modification to the xv6 code, which writes to the terminal each time the `log_write()` function is called. This function may be called from the following functions:

- `bzero`: Erases the contents of a block.
- `balloc`: Allocates a disk block to a file.
- `bfree`: Frees a disk block.
- `ialloc`: Allocates an inode when a file is created.
- `iupdate`: Updates the contents of an inode - very frequent.
- `bmap`: Not used.
- `writei`: Writes to a disk block - very frequent.

2. SINGLE FILE

```

$ echo > a1
log_write 34
log_write 34
log_write 59
$ echo x > a1
log_write 58
log_write 639
log_write 639
log_write 34
log_write 639
log_write 34
$ echo xxx > a1
log_write 639
log_write 639
log_write 639
log_write 34
log_write 639
log_write 34
$ rm a1
log_write 59
log_write 34
log_write 58
log_write 34
log_write 34
$ echo y > a2
log_write 34
log_write 34
log_write 59
log_write 58
log_write 639
log_write 639
log_write 34
log_write 639
log_write 34
$ █

```

Explanations for each command:

- (1) Since a new file is created, the inode (34) is allocated with `ialloc()` and updated with `iupdate()`, and the directory (59) is written to with `writei()`.
- (2) The free list - located at block 58 - is updated, and a block (639) is allocated to the new file within `balloc()`. The contents of block 639 are zeroed using `bzero()`.

Then we write data to the block (639 - once per character + 1 for newline) and update the inode (34) each time the size increases - each increment in size results in one `iupdate()` call and each character written to disk results in one `writei()` call.

Here we have two writes, increasing the size from 0 to 2, so there are two logs each for blocks 34 and 639 due to `iupdate()` and `writei()` respectively.

- (3) Since the size increases from 2 to 4 (x\n to xxx\n) and we write four characters to disk, we have 4 `writei()` calls for block 639 and 2 `iupdate()` calls for block 34.
- (4) The `rm` command unlinks the file, calling `sys_unlink()`. It updates the directory (59), removing the entry corresponding to `a1` with a `writei()`.

It also decrements the number of links to `a1` followed by a `iupdate()` call.

If the number of links to the file becomes 0, then the file is truncated and the blocks are appended to the free list - `bfree()` is called for each block within an `itrunc()` call.

Then, the inode is detached from the file by setting the type to 0 - this is followed by a `iupdate()` call. Then, it is invalidated by setting the valid field to 0.

- (5) The logs are simply the concatenated logs of 1 and 2.

3. MULTIPLE FILES

```
$ echo x > a1
log_write 34
log_write 34
log_write 59
log_write 58
log_write 639
log_write 639
log_write 34
log_write 639
log_write 34
$ echo y > a2
log_write 34
log_write 34
log_write 59
log_write 58
log_write 640
log_write 640
log_write 34
log_write 640
log_write 34
$ echo z > a3
log_write 34
log_write 34
log_write 59
log_write 58
log_write 641
log_write 641
log_write 34
log_write 641
log_write 34
$ rm a1 a2 a3
log_write 59
log_write 34
log_write 58
log_write 34
log_write 34
log_write 59
log_write 34
log_write 58
log_write 34
log_write 34
log_write 59
log_write 34
log_write 58
log_write 34
log_write 34
```

Explanations:

- (1) Each of the first three are identical to point 5 in the previous list, except the block number is incremented: 639, 640, 641, as they get popped off the free list.

The mechanism of each command is identical to the previous list.

- (2) The removal is also straightforward, and can be viewed as three iterations of point 4 in the previous list.