<u>File Systems</u>

Tanmay Goyal

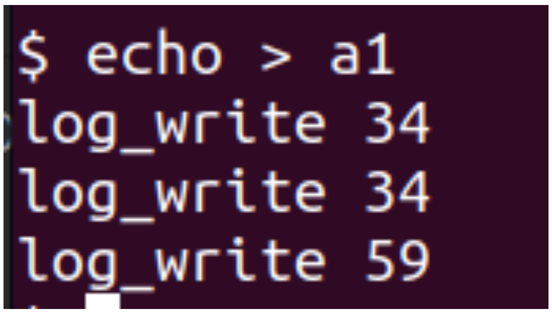
AI20BTECH11021

<u>Part 0</u>

- create(): used to create a new file or folder. Checks for the inode of the directory and if the file already exists, else allocates a new inode. It then links the file to the directory in case the new object is a file else it links it to '.'and '..'if it is a folder.
- sys_unlink(): used to unlink a file from a folder. It cannot unlink '.'or '..' from a folder.
- readi(): reads data from a inode
- writei(): writes data to an inode
- dirlink(): Enters the new name and inode into the directory.
- ialloc(): allocates an inode and marks the block as allocated
- iupdate(): updates the modified inode to the disk.
- iget(): finds the in memory copy for a inode
- ilock(): locks an inode for updating it.
- iunlock(): unlocks a given inode
- iput(): frees the inode and its content on the disk
- itrunc(): truncates an inode and discards its events
- bget(): looks for a buffer cache.
- bread(): returns the buffer with contents of the block
- bwrite(): writes content to the disk
- brelse(): releases a locked buffer

Part 1

When we run the following command *$ echo > a1* , we get the following output:

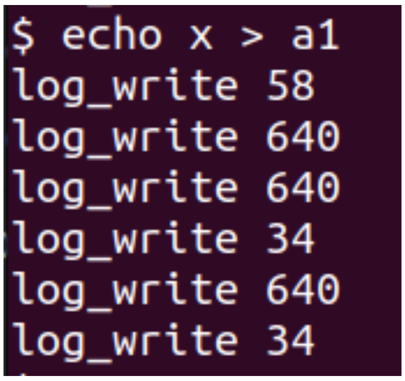


```
$ echo > a1
log_write 34
log_write 34
log_write 59
```

This is explained by the following:

First, the *create()* function is called. Within this, we first get the directory inode using *nameiparent()* and *namex()* function. It then makes sure there is no existing file of the same name by checking for file inodes. In case no such file exists, it calls upon *ialloc()* which marks block 34 as allocated and a referenced inode. It then calls upon *iupdate()* that marks the same block 34 as allocated on the disk. Finally, a call is made to *dirlink()* which links the file to the directory by adding the entry (name,inode) into the directory and calls upon *writei()*, which writes to block 59, which is the allocated buffer.

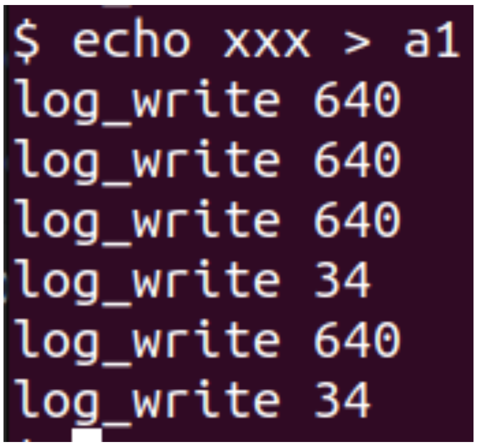
We now run *$ echo x > a1* , which results in the following output:



```
$ echo x > a1
log_write 58
log_write 640
log_write 640
log_write 34
log_write 640
log_write 34
```

First, the *create()* function is called. This time however, due to the file already existing, we donot need to call upon *ialloc()* and *iupdate()*. It then calls *balloc()*, which allocates a zeroed disk block, which in this case is Block 58, and writes this to the log. It then calls upon *bzero(),* which zeros the disk block 640, and this is followed by a *writei()* call to the block 640, writing the character “x” to the block and a call to *iupdate()* to update the inode for the file, stored at block 34, and then another set of calls to *writei()* and *iupdate()* for the EOF character.

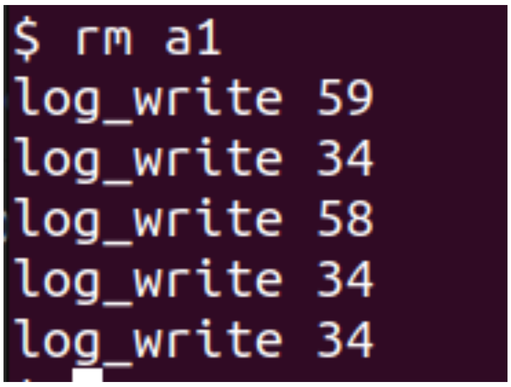
We now run *$ echo xxx > a1 ,* which results in the following output:



```
$ echo xxx > a1
log_write 640
log_write 640
log_write 640
log_write 34
log_write 640
log_write 34
```

Yes, there is a slight difference. This time there is no need to allocate a block and zero out a block since the zeroed-out block already exists. Thus, after the *create()* function, it straight up calls *writei()* three times to write the three "x" characters and then calls *iupdate()* to update the inode for the file, stored at block 34, and is followed by another set of calls to *writei()* and *iupdate()* for the EOF character.

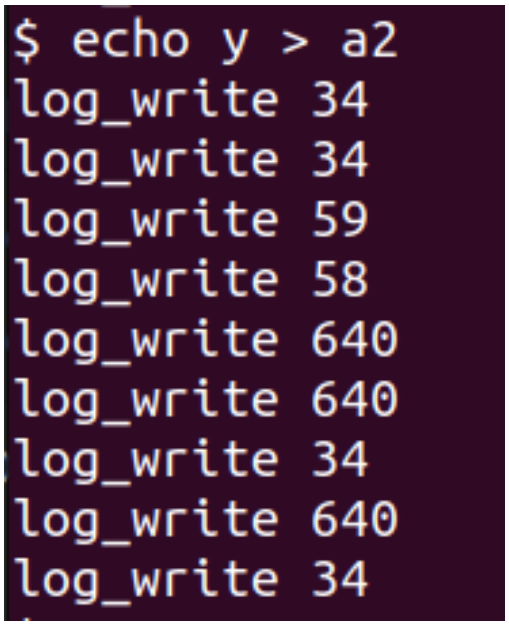
We now run *$ rm a1 ,* which results in the following output:



```
$ rm a1
log_write 59
log_write 34
log_write 58
log_write 34
log_write 34
```

The first call is to *sys_unlink()*, which calls *writei()* to write the original buffer 59 and updates the inode at block 34 using *iupdate()*. It then frees the allocated block 58 using *bfree()*. Then there are two calls to *iupdate()*, one is by *itrunc()* and one is by *iput()*. *Itrunc()* is repsonsivble for discarding the contents of the inode while *iput()* frees the inode and its content on the disk.

We now run *$ echo y > a2* , which results in the following output:



```
$ 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
```

The output is similar to the combined output of *$ echo > a1* and *$ echo x > a1. The create()* function is invoked which then looks for the inode of the directory and the inode of the file. Since it is a new file, a new block is allocated at block 34 for the inode and then it is also written to the disk. It then calls upon *dirlink()* to link the file to the directory. It then also allocates block 58 using *balloc()* and zeros block 641 using *bzero().* This is followed by a *writei()* call to the block 640, writing the character "x" to the block and a call to *iupdate()* to update the inode for the file, stored at block 34, and then another set of calls to *writei()* and *iupdate()* for the EOF character.

Part 2

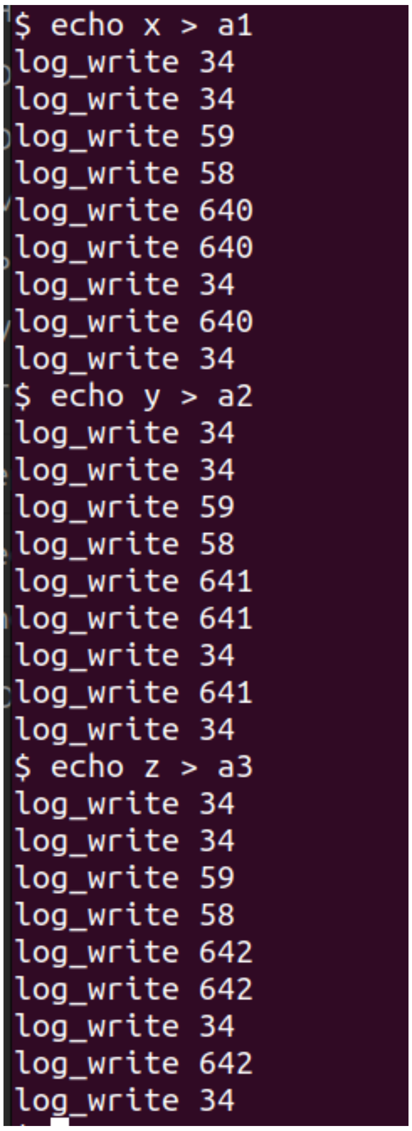
When we run the following sets of commands:

$ echo x > a1

$ echo y > a2

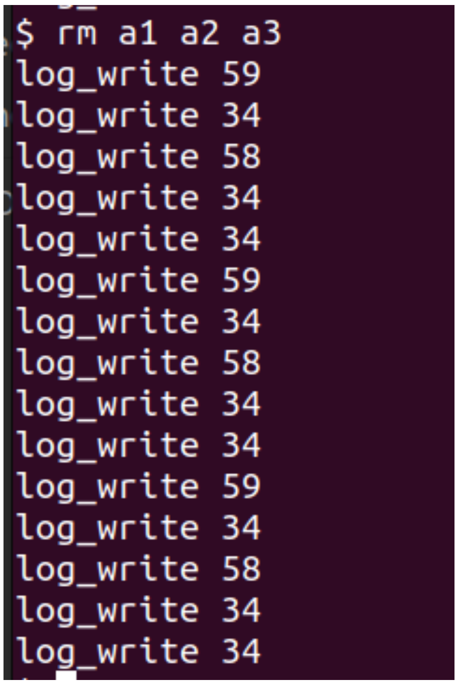
$ echo z > a3

We get the following output:


```
$ echo x > a1
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 y > a2
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
$ echo z > a3
log_write 34
log_write 34
log_write 59
log_write 58
log_write 642
log_write 642
log_write 34
log_write 642
log_write 34
```

For all the commands, the same procedure is being followed: we create a new file using *create()*, which checks upon if there is a inode corresponding to the directory and the file already, else it is allotted an inode. For all three files, the same block is being allotted to store the inodes, which is block 34. After this, *balloc()* and *bzero()* are being called to allocate a free disk block for each file. This is where the difference is being seen. *Balloc()* allocates block 58 for all three files, however, each file gets it's own zeroed-out block for its content. The first block gets block 640, the second gets 641 and the third gets 642. After that, there are two calls to the *writei()* function and *iupdate()* function, for the one character and the EOF character to be written to the separate blocks for each file, and to update the inodes for each file.

We now run *$ rm a1 a2 a3* , which results in the following output:



```
$ 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
```

Again, it is similar to the removal of one file. There is an invocation to *sys_unlink()* which calls *writei()* to write the original buffer 59 and updates the inode at block 34 using *iupdate()*. It then frees the allocated block 58 using *bfree()*. Then there are two calls to *iupdate()*, one is by *itrunc()* and one is by *iput()*. *Itrunc()* is responsible for discarding the contents of the inode while *iput()* frees the inode and its content on the disk. This same procedure is repeated for all three files.