EEL5737 Assignemnt 3

Sagnik Ghosh (sagnik.ghosh@ufl.edu)

November 2, 2020

Question 1.

The block server(*memoryfs_server.py*) holds the file system data. This has only 3 functions, put, get, and ReadSetBlock. These are accessed from the client side using XMLRPC. We can think these as the server side RPC stub. The first block in our file system is used to implement the ReadAndSetMemory. The various commands for accessing the file system, first reads and sets this memory to **LOCKED** value using **Acquire**. Meanwhile other threads which wants to access the memory will eventually wait for this memory to have **UNLOCKED** value. Once a client is done, it uses **Release** to set this value to **UNLOCKED**.

Value for LOCKED = bytearray of 1s

Value for UNLOCKED = bytearray of 0s

Question 2.

The File system client (*memoryfsclient.py*) remains exactly the same. Only difference is that the put and get methods now call the server over RPC to write data to block server. The lower level DiskBlocks goes distributed across the RPC, but the layers on top of it, Inode, InodeNumber, FileName remains exactly same.

Question 3.

When two threads tries to access the same memory location, and one of them is a "write" operation, this leads to the situation whoever wins the race, the other is affected by the operation of it. Different Race Conditions:

1. One Client issues a cat statement, the other issues an append statement, both on the same file. The result of the cat statement will depend on the execution of the append statement, if it's executed before or after the append.

2. Two clients simultaneously running a append statement, the outcome of the append depends on the order the two append statement was executed.

3. Two clients simultaneously running an append statement, with appends having more that one blocks to write to disk. The result in this case is unpredictable.

Observance Of Race Condition

I added a time.sleep(4) statement in the Write function of File Name class. Inside the while block after a data block is written.

Listing 1: Python example

```
block = file_inode.RawBlocks.Get(block_number)

# copy slice of data into the right position in the block
block[write_start:write_end] = data[bytes_written:bytes_written + (write_end -
write_start)]
```

```
# now write modified block back to disk
file_inode.RawBlocks.Put(block_number, block)
time.sleep(4)
# update offset, bytes written
current_offset += write_end - write_start
bytes_written += write_end - write_start
```

From one terminal, I ran the command.

```
append linktest.txt Sachin_has_two_centuries_he_is_the_greatest_batsman_of_all_time.
No_one_has_any_doubt_in_it.Those_who_does_not_agree_denies_GOD.He_is_inborn_talent_
and_he_works_hard_everyday_to_better_himself.
```

Then the second terminal, I ran the command to append to the same file.

```
append linktest.txt I_agree_not.RahulDravid_is_much_more_hard_working.
He_is_the_greatest_of_all_time.
```

This resulted in a race condition, and the second command overwrite the 1st block written by first append command. The content of the file appeared to be like the one below.

```
[cwd=0]:append linktest.txt Sachin_has_two_centuries_he_is_the_greatest_batsman_of_all_time.No_one_has_any_doubt_in_it.Those_who_does_not_agree_denies_GOD.He_is_inborn_talent_and_he_works_hard_everyday_to_better_himself.
Bytes Written: 192

[cwd=0]:cat linktest.txt
Contents of file9.txtI_agree_not.RahulDravid_is_much_more_hard_working.He_is_the_greatest_of_all_time.ubt_in_it.Those_who_does_not_agree_denies_GOD.He_is_inborn_talent_and_he_works_hard_everyday_to_better_himself.
[cwd=0]:
```

Figure 1: The final result

I introduced Acquire and Release on the shell level, therefore, guaranteeing the fact that only one user at a time can have access to the several layers of our file system. In this case, the sleep instruction doesn't cause a race condition. When one thread goes to sleep, the other threads won't be able to access the locks.

Describe how in your design you could keep track of access/modification times of files, and how that information could be used to support a client-side cache

This information can be included in the Inode layer of the file system. The Inode layer already has fields, type, size, and refcnt. We can introduce one more field as last modification/access time. Whenever the Inode is accessed, we can update this field with the timestamp of the request. Having this in both server and client side, we can eventually decide when cache is not coherent with the server, and thus delaying writes to server or serving request from cache and not accessing the server depending on the timestamp!