EEL 5737 Principles of Computer System Design

Final Project

UFID: 96198475

Name: Zezhou Zhang

Implementation

All the Functions of My Final Project:

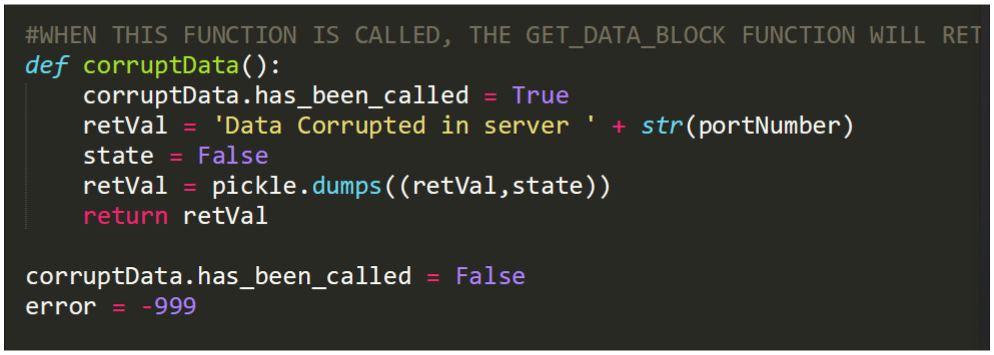
1. RAID-5 function in filesystem, which can allow random 1 server fail-stop.
2. RAID-5 function in filesystem, which can allow recovering the data block for read function after corrupting data block in one disk.
3. RAID-1 function in filesystem, which can allow random 1 server fail-stop.
4. RAID-1 function in filesystem, which can allow recovering the data block for read function after corrupting data block in one disk.

The Implementation of Server fail-stop Emulation:

Simply close one server port terminal (eg. Close port 8000)

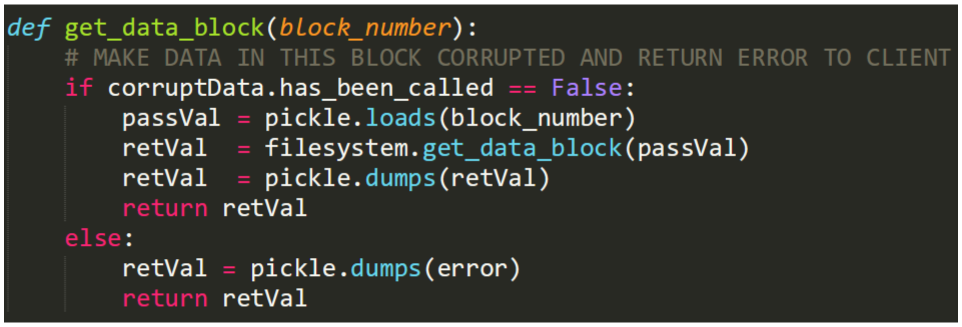
The Implementation of corrupted data:

After running backChannel.py, you can select 0, 1, 2, 3 to corrupt server. It will call the function: *corruptData()* in that corresponding proxy.



```python
#WHEN THIS FUNCTION IS CALLED, THE GET_DATA_BLOCK FUNCTION WILL RET
def corruptData():
    corruptData.has_been_called = True
    retVal = 'Data Corrupted in server ' + str(portNumber)
    state = False
    retVal = pickle.dumps((retVal,state))
    return retVal

corruptData.has_been_called = False
error = -999
```

This function can make *get_data_block()* return error when it is called:

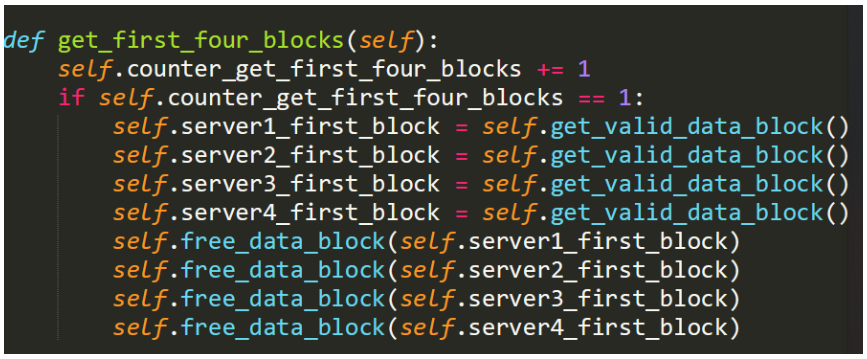


```python
def get_data_block(block_number):
    # MAKE DATA IN THIS BLOCK CORRUPTED AND RETURN ERROR TO CLIENT
    if corruptData.has_been_called == False:
        passVal = pickle.loads(block_number)
        retVal  = filesystem.get_data_block(passVal)
        retVal  = pickle.dumps(retVal)
        return retVal
    else:
        retVal = pickle.dumps(error)
        return retVal
```

The Implementation of Virtual-to-Physical Block Translation:

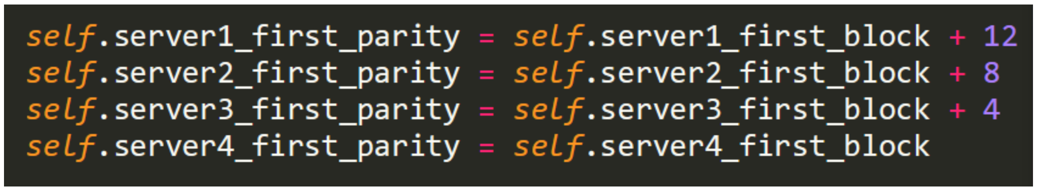
In RAID5:

First, in client_stub.py, I make InodeLayer.py run this function only one time to get first four blocks.



```python
def get_first_four_blocks(self):
    self.counter_get_first_four_blocks += 1
    if self.counter_get_first_four_blocks == 1:
        self.server1_first_block = self.get_valid_data_block()
        self.server2_first_block = self.get_valid_data_block()
        self.server3_first_block = self.get_valid_data_block()
        self.server4_first_block = self.get_valid_data_block()
        self.free_data_block(self.server1_first_block)
        self.free_data_block(self.server2_first_block)
        self.free_data_block(self.server3_first_block)
        self.free_data_block(self.server4_first_block)
```

Second, calculate the first parity block number for 4 servers.



```python
self.server1_first_parity = self.server1_first_block + 12
self.server2_first_parity = self.server2_first_block + 8
self.server3_first_parity = self.server3_first_block + 4
self.server4_first_parity = self.server4_first_block
```

Third, implement the translation function:

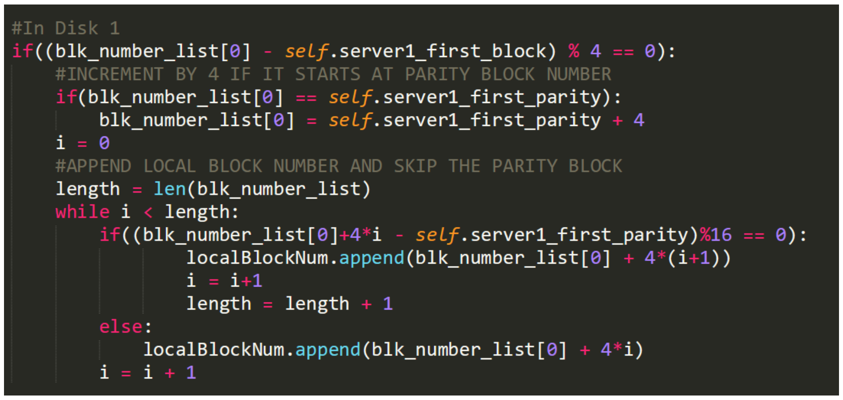
The large file in one inode should be store in one disk. For example, in only 1 server case, the large file will use 23, 24, 25, 26, 27, 28 blocks (virtual block numbers).

23	24	25	26
27	28	29	30

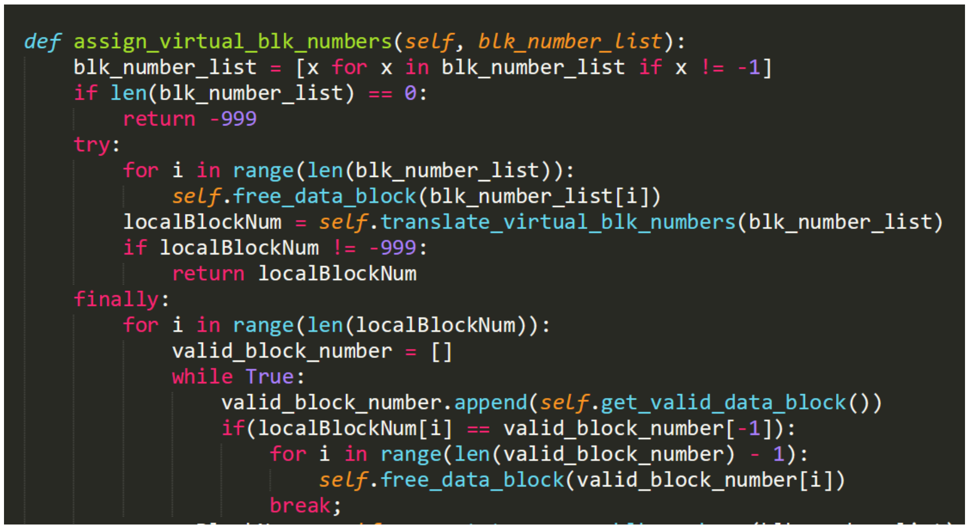
But in RAID5, the large file will use 23, 27, 31, 39, 43, 47 physical block numbers, and it should be stored like that:

🟦 23	24	25	🟥 26
🟦 27	28	🟥 29	30
🟦 31	🟥 32	33	34
🟥 35	36	37	38
🟦 39	40	41	🟥 42
🟦 43	44	🟥 45	46
🟦 47	🟥 48	49	50
🟥			

So that, in this example, use server1_first_block to locate which server is going to use for storing the data, if it starts at 39, then (39% server1_first_block) is equal to 0. But if it starts at parity block number, it should increment by 4 and leave the party block blank. After that, I use the server1_first_parity to locate all the parity block numbers in Disk one. Then, based on the length, it can occupy the data block in Disk one but skip the parity block.



```
#In Disk 1
if((blk_number_list[0] - self.server1_first_block) % 4 == 0):
    #INCREMENT BY 4 IF IT STARTS AT PARITY BLOCK NUMBER
    if(blk_number_list[0] == self.server1_first_parity):
        blk_number_list[0] = self.server1_first_parity + 4
    i = 0
    #APPEND LOCAL BLOCK NUMBER AND SKIP THE PARITY BLOCK
    length = len(blk_number_list)
    while i < length:
        if((blk_number_list[0]+4*i - self.server1_first_parity)%16 == 0):
                localBlockNum.append(blk_number_list[0] + 4*(i+1))
                i = i+1
                length = length + 1
        else:
            localBlockNum.append(blk_number_list[0] + 4*i)
        i = i + 1
```



```python
def assign_virtual_blk_numbers(self, blk_number_list):
    blk_number_list = [x for x in blk_number_list if x != -1]
    if len(blk_number_list) == 0:
        return -999
    try:
        for i in range(len(blk_number_list)):
            self.free_data_block(blk_number_list[i])
        localBlockNum = self.translate_virtual_blk_numbers(blk_number_list)
        if localBlockNum != -999:
            return localBlockNum
    finally:
        for i in range(len(localBlockNum)):
            valid_block_number = []
            while True:
                valid_block_number.append(self.get_valid_data_block())
                if(localBlockNum[i] == valid_block_number[-1]):
                    for i in range(len(valid_block_number) - 1):
                        self.free_data_block(valid_block_number[i])
                    break;
```

However, due to the characteristic of *get_valid_data_block()* and *free_data_block()*, we have to implement a function to accomplish the virtual-pysical translation. The *get_valid_data_block()* in memory.py will only give you the data block in series in the original filesystem layout. In order to get the valid data block in the wanted disk, we must keep call *get_valid_data_block()* until we find the designated block numbers. And use *free_data_block()* to free the blocks that are on the way we look for the designated blocks.

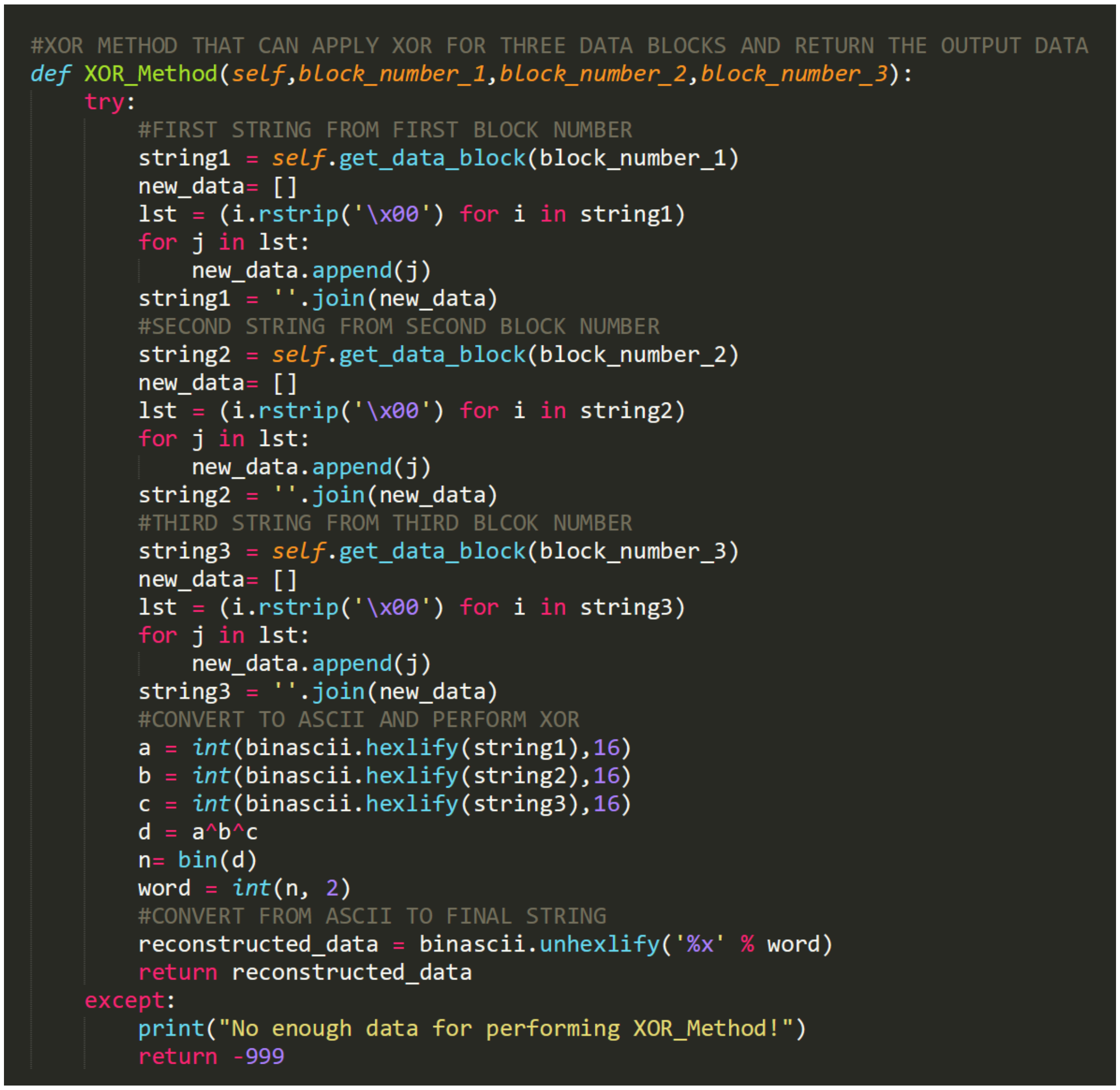
23 FIND	24 FREE	25 FREE	26 FREE
27 FIND	28	29	30

23	24	25	26
27	28	29	30

Eg. Find physical block 23, 27 by getting 23, 24, 25, 26, 27 and free 24, 25, 26, 27

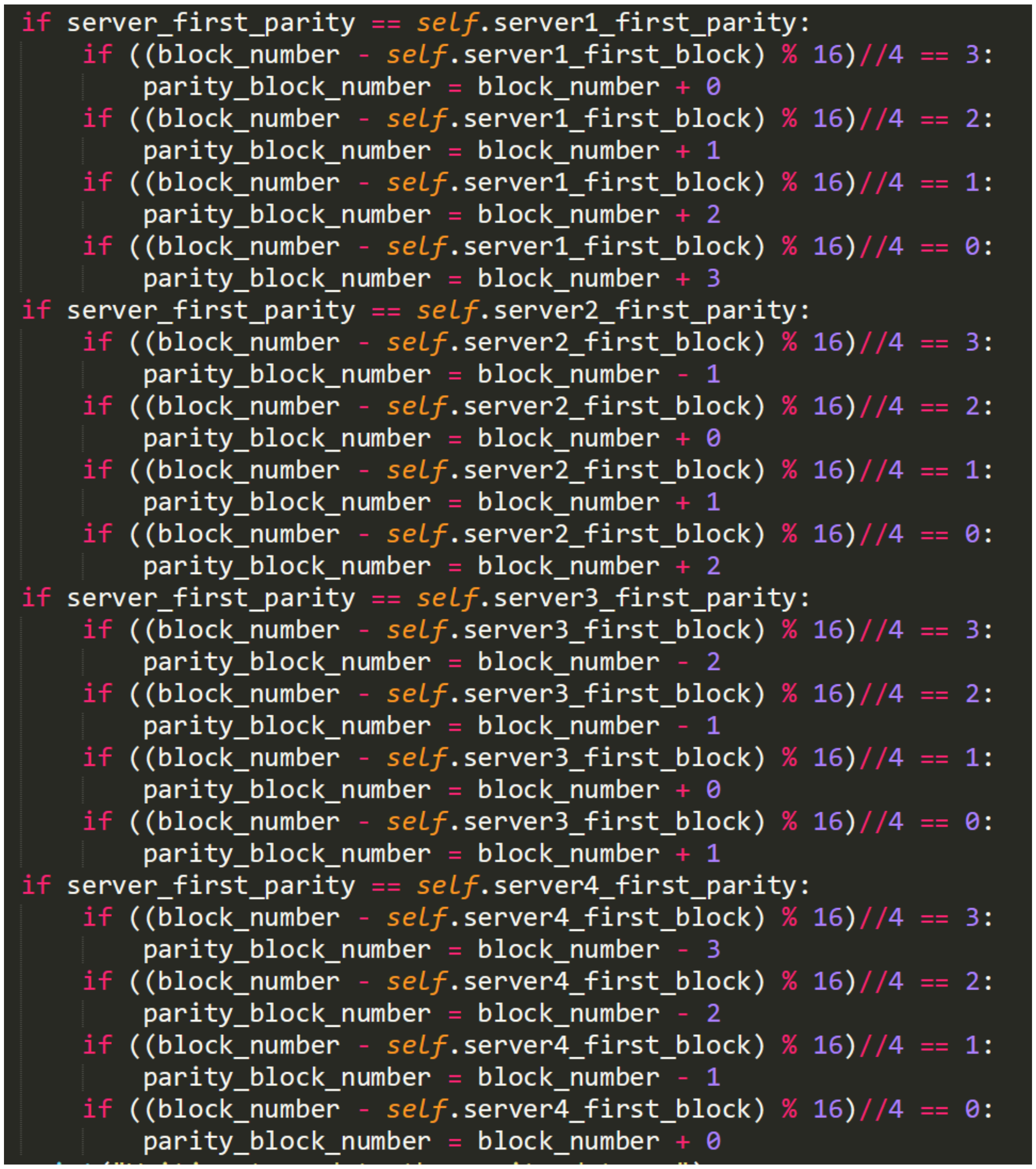
The Implementation of Updating Parity Block:

First of all, we need to implement the XOR-Method, which is the foundation to create parity data and recovery of data block during server fail-stop and corruption.

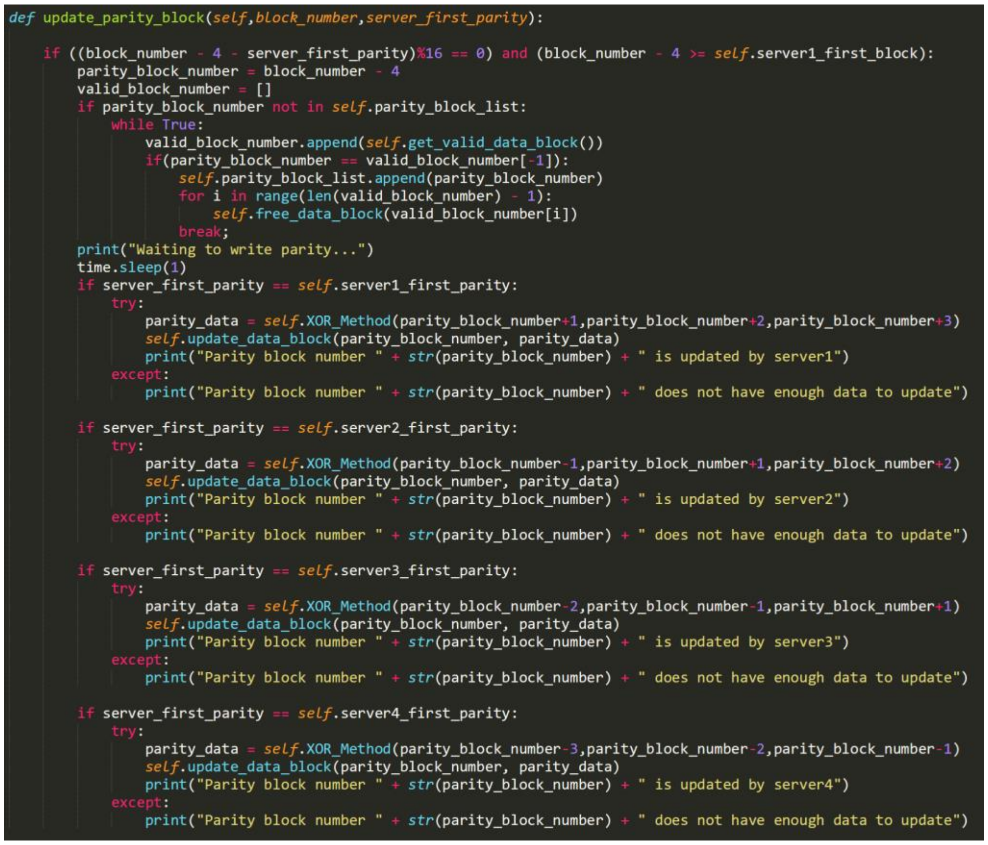


```
#XOR METHOD THAT CAN APPLY XOR FOR THREE DATA BLOCKS AND RETURN THE OUTPUT DATA
def XOR_Method(self,block_number_1,block_number_2,block_number_3):
    try:
        #FIRST STRING FROM FIRST BLOCK NUMBER
        string1 = self.get_data_block(block_number_1)
        new_data= []
        lst = (i.rstrip('\x00') for i in string1)
        for j in lst:
            new_data.append(j)
        string1 = ''.join(new_data)
        #SECOND STRING FROM SECOND BLOCK NUMBER
        string2 = self.get_data_block(block_number_2)
        new_data= []
        lst = (i.rstrip('\x00') for i in string2)
        for j in lst:
            new_data.append(j)
        string2 = ''.join(new_data)
        #THIRD STRING FROM THIRD BLCOK NUMBER
        string3 = self.get_data_block(block_number_3)
        new_data= []
        lst = (i.rstrip('\x00') for i in string3)
        for j in lst:
            new_data.append(j)
        string3 = ''.join(new_data)
        #CONVERT TO ASCII AND PERFORM XOR
        a = int(binascii.hexlify(string1),16)
        b = int(binascii.hexlify(string2),16)
        c = int(binascii.hexlify(string3),16)
        d = a^b^c
        n= bin(d)
        word = int(n, 2)
        #CONVERT FROM ASCII TO FINAL STRING
        reconstructed_data = binascii.unhexlify('%x' % word)
        return reconstructed_data
    except:
        print("No enough data for performing XOR_Method!")
        return -999
```

After that, I use *attempt_to_update_parity_block()* to try to update parity block before every time updating the data block by using XOR(new data, old parity, and old data) to update the parity block. It is possible that parity block is empty because it is not updated due to not enough data on other 3 blocks. So it will ‘try’ to *get_data_block()*, and print the error when it failed to get data block.


```
if server_first_parity == self.server1_first_parity:
    if ((block_number - self.server1_first_block) % 16)//4 == 3:
        parity_block_number = block_number + 0
    if ((block_number - self.server1_first_block) % 16)//4 == 2:
        parity_block_number = block_number + 1
    if ((block_number - self.server1_first_block) % 16)//4 == 1:
        parity_block_number = block_number + 2
    if ((block_number - self.server1_first_block) % 16)//4 == 0:
        parity_block_number = block_number + 3
if server_first_parity == self.server2_first_parity:
    if ((block_number - self.server2_first_block) % 16)//4 == 3:
        parity_block_number = block_number - 1
    if ((block_number - self.server2_first_block) % 16)//4 == 2:
        parity_block_number = block_number + 0
    if ((block_number - self.server2_first_block) % 16)//4 == 1:
        parity_block_number = block_number + 1
    if ((block_number - self.server2_first_block) % 16)//4 == 0:
        parity_block_number = block_number + 2
if server_first_parity == self.server3_first_parity:
    if ((block_number - self.server3_first_block) % 16)//4 == 3:
        parity_block_number = block_number - 2
    if ((block_number - self.server3_first_block) % 16)//4 == 2:
        parity_block_number = block_number - 1
    if ((block_number - self.server3_first_block) % 16)//4 == 1:
        parity_block_number = block_number + 0
    if ((block_number - self.server3_first_block) % 16)//4 == 0:
        parity_block_number = block_number + 1
if server_first_parity == self.server4_first_parity:
    if ((block_number - self.server4_first_block) % 16)//4 == 3:
        parity_block_number = block_number - 3
    if ((block_number - self.server4_first_block) % 16)//4 == 2:
        parity_block_number = block_number - 2
    if ((block_number - self.server4_first_block) % 16)//4 == 1:
        parity_block_number = block_number - 1
    if ((block_number - self.server4_first_block) % 16)//4 == 0:
        parity_block_number = block_number + 0
```

Above is showing how to know the location of parity block when knowing the updated data block number. It can show the parity block number in the row where the updated data block is.



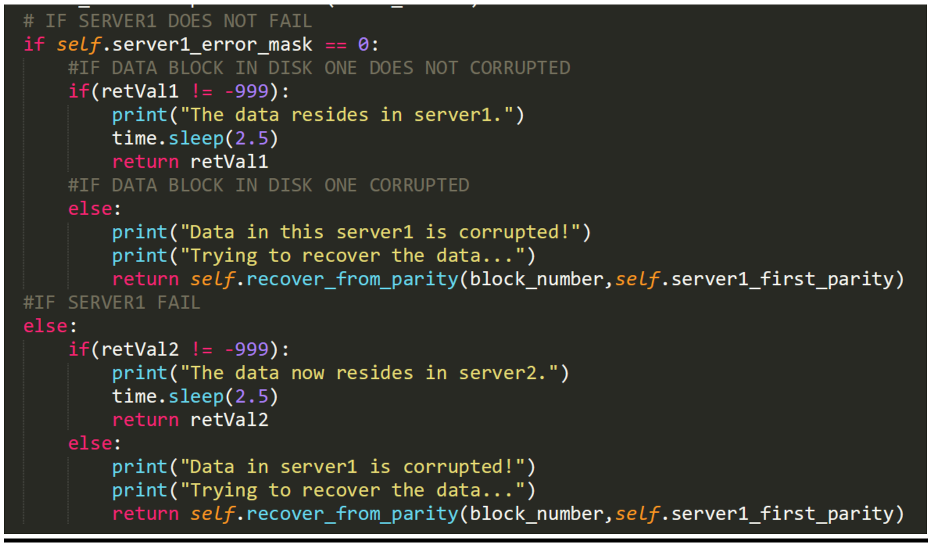
```python
def update_parity_block(self,block_number,server_first_parity):

    if ((block_number - 4 - server_first_parity)%16 == 0) and (block_number - 4 >= self.server1_first_block):
        parity_block_number = block_number - 4
        valid_block_number = []
        if parity_block_number not in self.parity_block_list:
            while True:
                valid_block_number.append(self.get_valid_data_block())
                if(parity_block_number == valid_block_number[-1]):
                    self.parity_block_list.append(parity_block_number)
                    for i in range(len(valid_block_number) - 1):
                        self.free_data_block(valid_block_number[i])
                    break;
        print("Waiting to write parity...")
        time.sleep(1)
        if server_first_parity == self.server1_first_parity:
            try:
                parity_data = self.XOR_Method(parity_block_number+1,parity_block_number+2,parity_block_number+3)
                self.update_data_block(parity_block_number, parity_data)
                print("Parity block number " + str(parity_block_number) + " is updated by server1")
            except:
                print("Parity block number " + str(parity_block_number) + " does not have enough data to update")

        if server_first_parity == self.server2_first_parity:
            try:
                parity_data = self.XOR_Method(parity_block_number-1,parity_block_number+1,parity_block_number+2)
                self.update_data_block(parity_block_number, parity_data)
                print("Parity block number " + str(parity_block_number) + " is updated by server2")
            except:
                print("Parity block number " + str(parity_block_number) + " does not have enough data to update")

        if server_first_parity == self.server3_first_parity:
            try:
                parity_data = self.XOR_Method(parity_block_number-2,parity_block_number-1,parity_block_number+1)
                self.update_data_block(parity_block_number, parity_data)
                print("Parity block number " + str(parity_block_number) + " is updated by server3")
            except:
                print("Parity block number " + str(parity_block_number) + " does not have enough data to update")

        if server_first_parity == self.server4_first_parity:
            try:
                parity_data = self.XOR_Method(parity_block_number-3,parity_block_number-2,parity_block_number-1)
                self.update_data_block(parity_block_number, parity_data)
                print("Parity block number " + str(parity_block_number) + " is updated by server4")
            except:
                print("Parity block number " + str(parity_block_number) + " does not have enough data to update")
```

After that, using *update_parity_block()* in the *updata_data_block()* to write parity data to the parity block if the block number above the updated data block is parity block number (block number - 4). The way how I implement is because it has high possibility that parity block can have data from the other 3 blocks if the filesystem begins to write data in the next row. So, it can update parity block without keeping raise error message for not enough data block. But the drawback is that the parity data block is not going to be updated as soon as the other 3 blocks around are all having data.

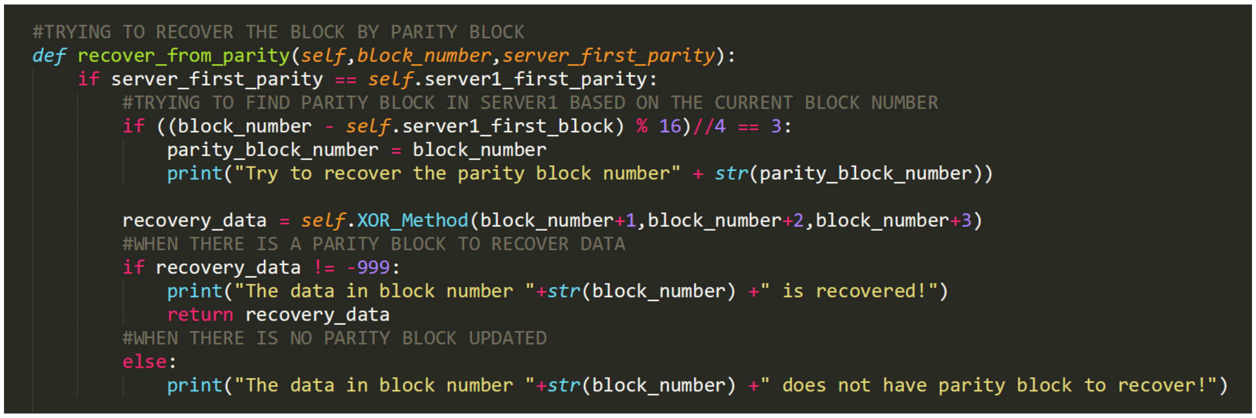
23	24	25	26
27	28	29	30
31	32	33	34

Eg. Update parity block 26 when block 30 is updated.

Implementation of Recovery of Data



```
# IF SERVER1 DOES NOT FAIL
if self.server1_error_mask == 0:
    #IF DATA BLOCK IN DISK ONE DOES NOT CORRUPTED
    if(retVal1 != -999):
        print("The data resides in server1.")
        time.sleep(2.5)
        return retVal1
    #IF DATA BLOCK IN DISK ONE CORRUPTED
    else:
        print("Data in this server1 is corrupted!")
        print("Trying to recover the data...")
        return self.recover_from_parity(block_number,self.server1_first_parity)
#IF SERVER1 FAIL
else:
    if(retVal2 != -999):
        print("The data now resides in server2.")
        time.sleep(2.5)
        return retVal2
    else:
        print("Data in server1 is corrupted!")
        print("Trying to recover the data...")
        return self.recover_from_parity(block_number,self.server1_first_parity)
```



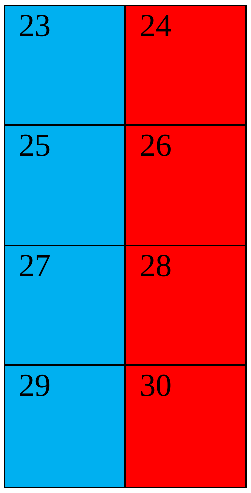
```
#TRYING TO RECOVER THE BLOCK BY PARITY BLOCK
def recover_from_parity(self,block_number,server_first_parity):
    if server_first_parity == self.server1_first_parity:
        #TRYING TO FIND PARITY BLOCK IN SERVER1 BASED ON THE CURRENT BLOCK NUMBER
        if ((block_number - self.server1_first_block) % 16)//4 == 3:
            parity_block_number = block_number
            print("Try to recover the parity block number" + str(parity_block_number))

        recovery_data = self.XOR_Method(block_number+1,block_number+2,block_number+3)
        #WHEN THERE IS A PARITY BLOCK TO RECOVER DATA
        if recovery_data != -999:
            print("The data in block number "+str(block_number) +" is recovered!")
            return recovery_data
        #WHEN THERE IS NO PARITY BLOCK UPDATED
        else:
            print("The data in block number "+str(block_number) +" does not have parity block to recover!")
```

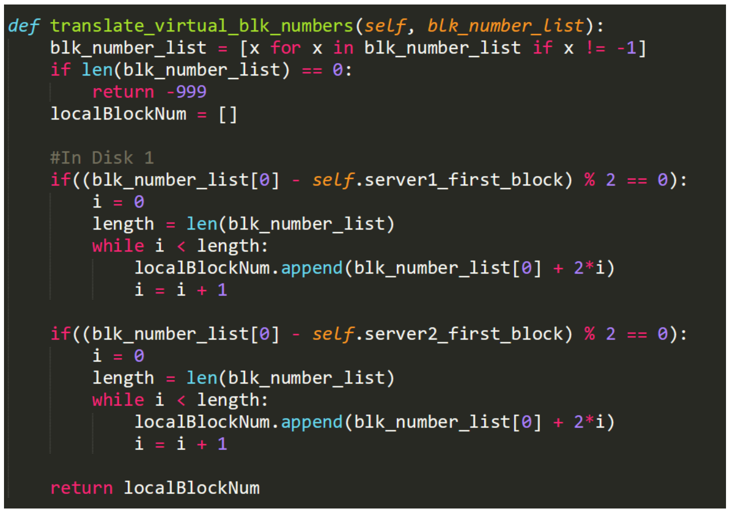
server1_error_mask is used to transfer the function to server2, so that it will not keep print server1 fail message every time *get_data_block()* is called. When the data block is corrupted in the server you choose, if there is a non-empty parity block exist, it will try to recover the data. Therefore, once a server failed, the error message will only appear once, and the other servers will take the commands.

In RAID1

Implemetation of Laying Out Blocks:

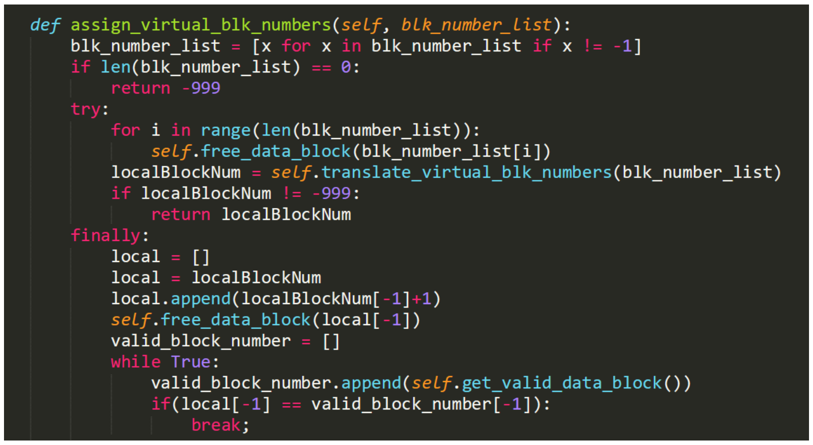
The RAID1 will copy the entire data blocks as replica in another disk, so the available storage can be a half of whole storage.





```
def translate_virtual_blk_numbers(self, blk_number_list):
    blk_number_list = [x for x in blk_number_list if x != -1]
    if len(blk_number_list) == 0:
        return -999
    localBlockNum = []

    #In Disk 1
    if((blk_number_list[0] - self.server1_first_block) % 2 == 0):
        i = 0
        length = len(blk_number_list)
        while i < length:
            localBlockNum.append(blk_number_list[0] + 2*i)
            i = i + 1

    if((blk_number_list[0] - self.server2_first_block) % 2 == 0):
        i = 0
        length = len(blk_number_list)
        while i < length:
            localBlockNum.append(blk_number_list[0] + 2*i)
            i = i + 1

    return localBlockNum
```

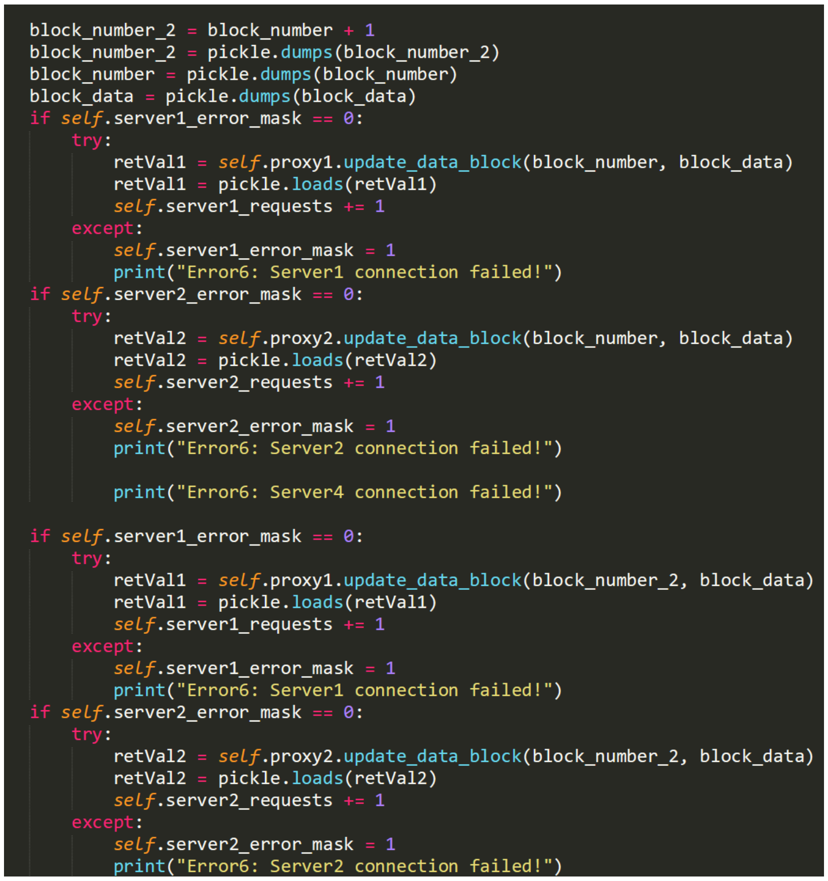
If the virtual block numbers are 23, 24, 25, 26, then the physical block number should be 23, 25, 27, 29. The implementation can be almost the same as RAID5.



```
def assign_virtual_blk_numbers(self, blk_number_list):
    blk_number_list = [x for x in blk_number_list if x != -1]
    if len(blk_number_list) == 0:
        return -999
    try:
        for i in range(len(blk_number_list)):
            self.free_data_block(blk_number_list[i])
        localBlockNum = self.translate_virtual_blk_numbers(blk_number_list)
        if localBlockNum != -999:
            return localBlockNum
    finally:
        local = []
        local = localBlockNum
        local.append(localBlockNum[-1]+1)
        self.free_data_block(local[-1])
        valid_block_number = []
        while True:
            valid_block_number.append(self.get_valid_data_block())
            if(local[-1] == valid_block_number[-1]):
                break;
```

However, the problem of *get_valid_data_block()* and *free_data_block()* still exists. Since every time one data block is updated, the block next to it should also be updated. So, after getting valid data block for 23, 25, 27, 29, we can also get one more valid block: block 30, and do not need to free data block in RAID1 since they should all be updated from 23-30.

Implemetation of Updating Data Blocks:

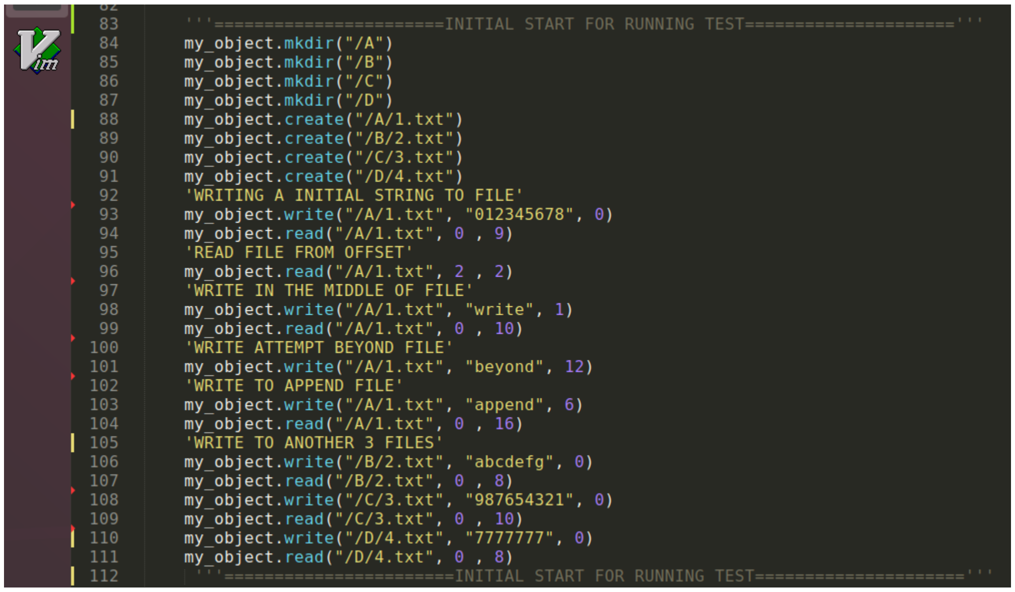


```python
block_number_2 = block_number + 1
block_number_2 = pickle.dumps(block_number_2)
block_number = pickle.dumps(block_number)
block_data = pickle.dumps(block_data)
if self.server1_error_mask == 0:
    try:
        retVal1 = self.proxy1.update_data_block(block_number, block_data)
        retVal1 = pickle.loads(retVal1)
        self.server1_requests += 1
    except:
        self.server1_error_mask = 1
        print("Error6: Server1 connection failed!")
if self.server2_error_mask == 0:
    try:
        retVal2 = self.proxy2.update_data_block(block_number, block_data)
        retVal2 = pickle.loads(retVal2)
        self.server2_requests += 1
    except:
        self.server2_error_mask = 1
        print("Error6: Server2 connection failed!")

        print("Error6: Server4 connection failed!")

if self.server1_error_mask == 0:
    try:
        retVal1 = self.proxy1.update_data_block(block_number_2, block_data)
        retVal1 = pickle.loads(retVal1)
        self.server1_requests += 1
    except:
        self.server1_error_mask = 1
        print("Error6: Server1 connection failed!")
if self.server2_error_mask == 0:
    try:
        retVal2 = self.proxy2.update_data_block(block_number_2, block_data)
        retVal2 = pickle.loads(retVal2)
        self.server2_requests += 1
    except:
        self.server2_error_mask = 1
        print("Error6: Server2 connection failed!")
```

Every time update the data block, the block next to it (block number + 1) should be updated as replica.

Step by Step Instruction:

Setting the initial start in FileSystem.py for testing (comment this area if want to type from beginning):



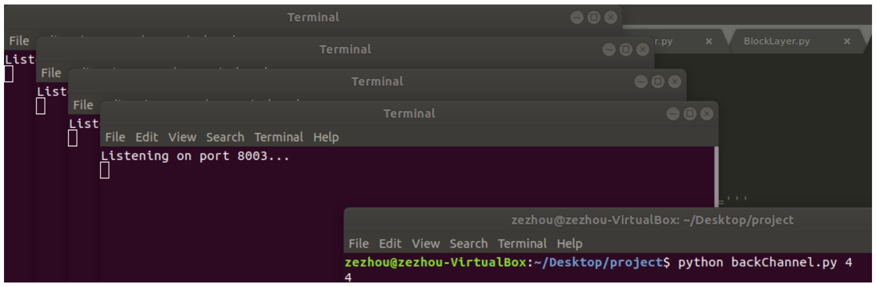
```
'''=======================INITIAL START FOR RUNNING TEST====================='''
my_object.mkdir("/A")
my_object.mkdir("/B")
my_object.mkdir("/C")
my_object.mkdir("/D")
my_object.create("/A/1.txt")
my_object.create("/B/2.txt")
my_object.create("/C/3.txt")
my_object.create("/D/4.txt")
'WRITING A INITIAL STRING TO FILE'
my_object.write("/A/1.txt", "012345678", 0)
my_object.read("/A/1.txt", 0 , 9)
'READ FILE FROM OFFSET'
my_object.read("/A/1.txt", 2 , 2)
'WRITE IN THE MIDDLE OF FILE'
my_object.write("/A/1.txt", "write", 1)
my_object.read("/A/1.txt", 0 , 10)
'WRITE ATTEMPT BEYOND FILE'
my_object.write("/A/1.txt", "beyond", 12)
'WRITE TO APPEND FILE'
my_object.write("/A/1.txt", "append", 6)
my_object.read("/A/1.txt", 0 , 16)
'WRITE TO ANOTHER 3 FILES'
my_object.write("/B/2.txt", "abcdefg", 0)
my_object.read("/B/2.txt", 0 , 8)
my_object.write("/C/3.txt", "987654321", 0)
my_object.read("/C/3.txt", 0 , 10)
my_object.write("/D/4.txt", "7777777", 0)
my_object.read("/D/4.txt", 0 , 8)
'''=======================INITIAL START FOR RUNNING TEST====================='''
```

<u>For testing the RAID-5 function with 1 server fail-stop:</u>

1) Open a terminal in Linux environment where all the python files are, and type:

python backChannel.py 4

where 4 means 4 servers, which can open 4 server ports: 8000, 8001, 8002, 8003

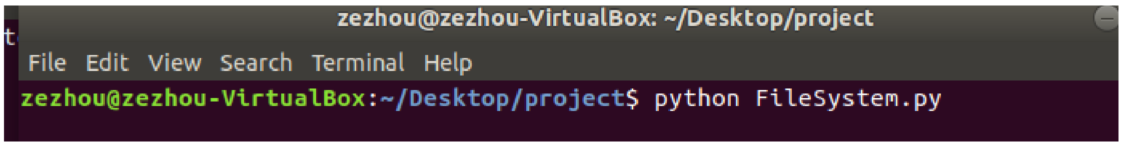


```
Listening on port 8003...
```

```
zezhou@zezhou-VirtualBox:~/Desktop/project$ python backChannel.py 4
4
```

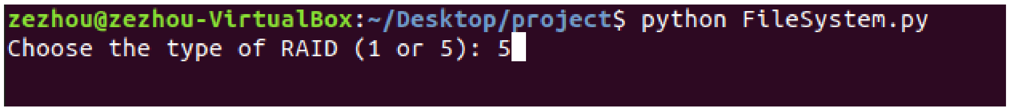
2) Open another terminal also where the python files are, and type:

python FileSystem.py



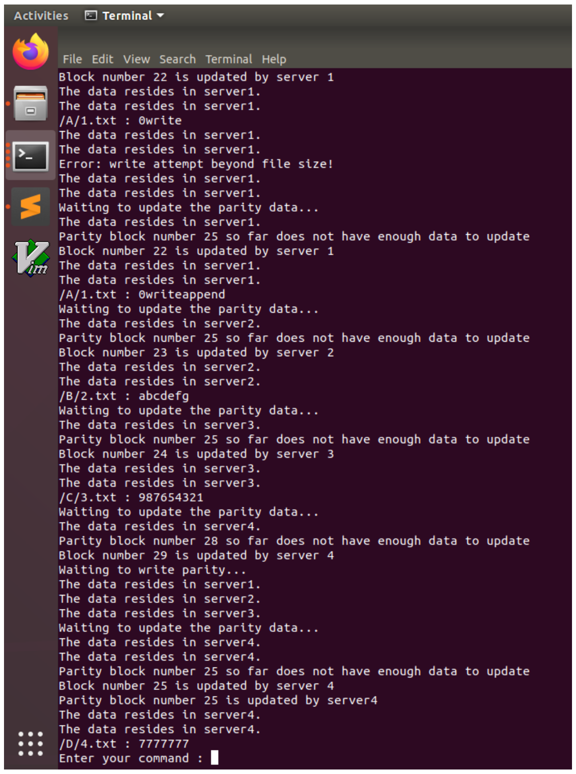
```
zezhou@zezhou-VirtualBox:~/Desktop/project$ python FileSystem.py
```

3) Choose which type of RAID you want to use (eg. Type ‘5’ for RAID5)



```
zezhou@zezhou-VirtualBox:~/Desktop/project$ python FileSystem.py
Choose the type of RAID (1 or 5): 5
```

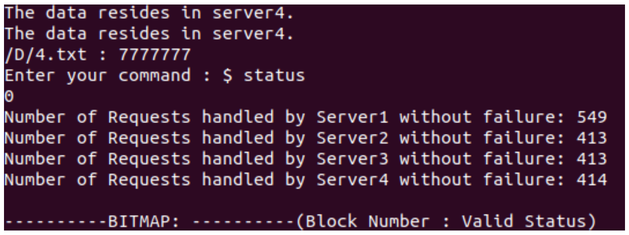
4) For ease the test process, use all the commands already in FileSystem.py:



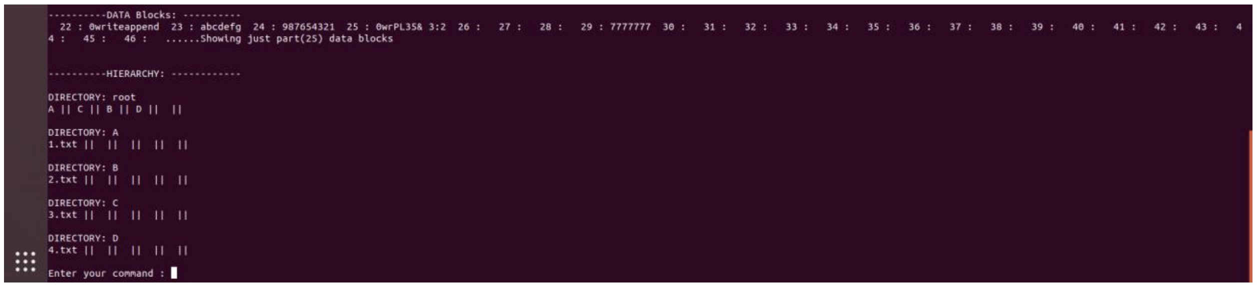
```
Block number 22 is updated by server 1
The data resides in server1.
The data resides in server1.
/A/1.txt : 0write
The data resides in server1.
The data resides in server1.
Error: write attempt beyond file size!
The data resides in server1.
The data resides in server1.
Waiting to update the parity data...
The data resides in server1.
Parity block number 25 so far does not have enough data to update
Block number 22 is updated by server 1
The data resides in server1.
The data resides in server1.
/A/1.txt : 0writeappend
Waiting to update the parity data...
The data resides in server2.
Parity block number 25 so far does not have enough data to update
Block number 23 is updated by server 2
The data resides in server2.
The data resides in server2.
/B/2.txt : abcdefg
Waiting to update the parity data...
The data resides in server3.
Parity block number 25 so far does not have enough data to update
Block number 24 is updated by server 3
The data resides in server3.
The data resides in server3.
/C/3.txt : 987654321
Waiting to update the parity data...
The data resides in server4.
Parity block number 28 so far does not have enough data to update
Block number 29 is updated by server 4
Waiting to write parity...
The data resides in server1.
The data resides in server2.
The data resides in server3.
Waiting to update the parity data...
The data resides in server4.
The data resides in server4.
Parity block number 25 so far does not have enough data to update
Block number 25 is updated by server 4
Parity block number 25 is updated by server4
The data resides in server4.
The data resides in server4.
/D/4.txt : 7777777
Enter your command :
```

5) See the status of filesystem and check if all blocks and parity blocks lay out like RAID5:

$ status



```
The data resides in server4.
The data resides in server4.
/D/4.txt : 7777777
Enter your command : $ status
0
Number of Requests handled by Server1 without failure: 549
Number of Requests handled by Server2 without failure: 413
Number of Requests handled by Server3 without failure: 413
Number of Requests handled by Server4 without failure: 414

----------BITMAP: ----------(Block Number : Valid Status)
```



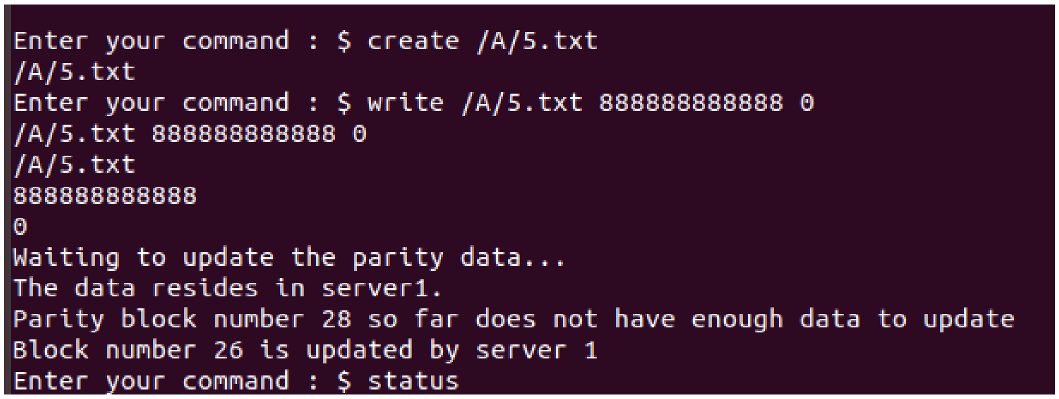
```
----------DATA Blocks: ----------
  22 : 0writeappend  23 : abcdefg  24 : 987654321  25 : 0wrPL35& 3:2  26 :   27 :   28 :   29 : 7777777  30 :   31 :   32 :   33 :   34 :   35 :   36 :   37 :   38 :   39 :   40 :   41 :   42 :   43 :   4
4 :   45 :   46 :   ......Showing just part(25) data blocks


----------HIERARCHY: -----------

DIRECTORY: root
A || C || B || D ||  ||

DIRECTORY: A
1.txt ||  ||  ||  ||  ||

DIRECTORY: B
2.txt ||  ||  ||  ||  ||

DIRECTORY: C
3.txt ||  ||  ||  ||  ||

DIRECTORY: D
4.txt ||  ||  ||  ||  ||

Enter your command :
```

6) Test the write function by type

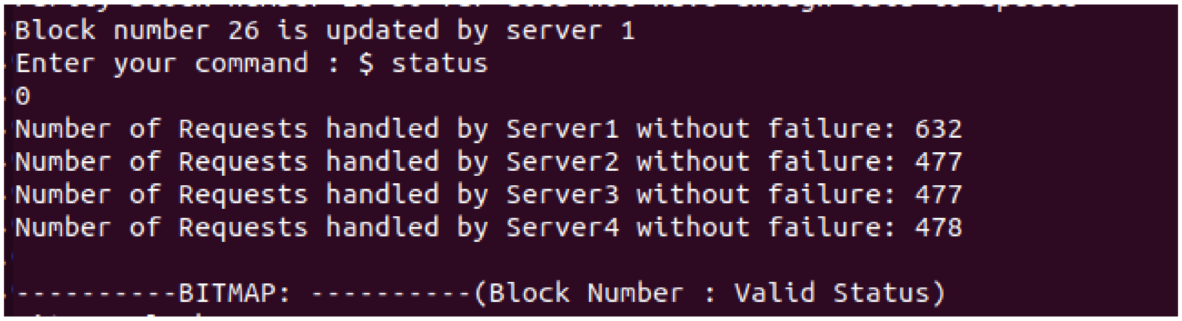
$ create /A/5.txt

$ write /A/5.txt 88888888888 0

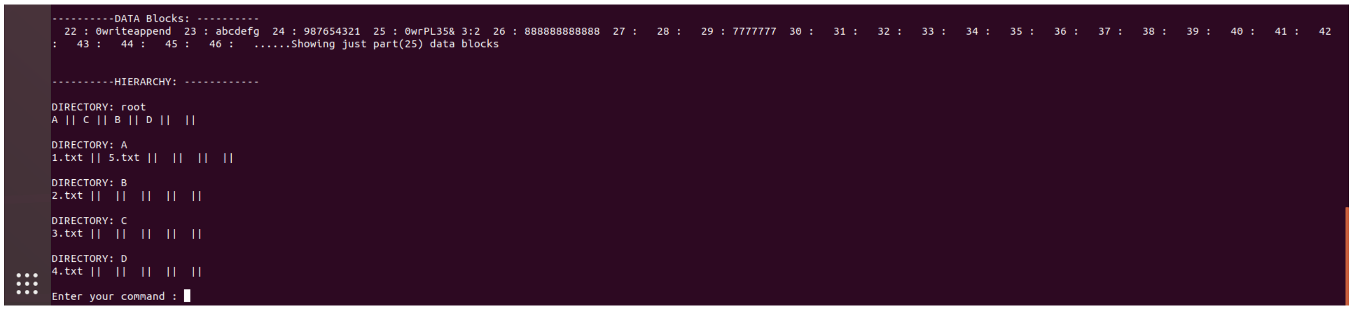
$ status



```
Enter your command : $ create /A/5.txt
/A/5.txt
Enter your command : $ write /A/5.txt 888888888888 0
/A/5.txt 888888888888 0
/A/5.txt
888888888888
0
Waiting to update the parity data...
The data resides in server1.
Parity block number 28 so far does not have enough data to update
Block number 26 is updated by server 1
Enter your command : $ status
```



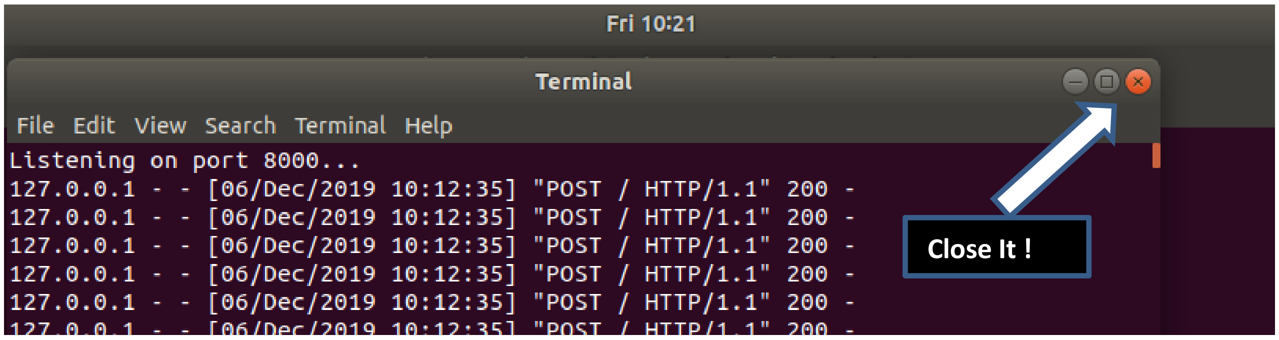
```
Block number 26 is updated by server 1
Enter your command : $ status
0
Number of Requests handled by Server1 without failure: 632
Number of Requests handled by Server2 without failure: 477
Number of Requests handled by Server3 without failure: 477
Number of Requests handled by Server4 without failure: 478

----------BITMAP: ----------(Block Number : Valid Status)
```



```
----------DATA Blocks: ----------
  22 : 0writeappend  23 : abcdefg  24 : 987654321  25 : 0wrPL35& 3:2  26 : 888888888888  27 :   28 :   29 : 7777777  30 :   31 :   32 :   33 :   34 :   35 :   36 :   37 :   38 :   39 :   40 :   41 :   42
:   43 :   44 :   45 :   46 :   ......Showing just part(25) data blocks


----------HIERARCHY: -----------

DIRECTORY: root
A || C || B || D ||  ||

DIRECTORY: A
1.txt || 5.txt ||  ||  ||  ||

DIRECTORY: B
2.txt ||  ||  ||  ||  ||

DIRECTORY: C
3.txt ||  ||  ||  ||  ||

DIRECTORY: D
4.txt ||  ||  ||  ||  ||

Enter your command :
```

22: 0writeappend */A/1.txt*	23: abcdefg */B/2.txt*	24: 987654321 */C/3.txt*	25: 0wrPL35& 3:2 *Parity Block 1*
26: 888888888888 */A/5.txt*	27:	28:	29:7777777 */D/4.txt*

7) Simply close one terminal to emulate 1 server fail-stop (eg. Close port 8000)



```
Listening on port 8000...
127.0.0.1 - - [06/Dec/2019 10:12:35] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [06/Dec/2019 10:12:35] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [06/Dec/2019 10:12:35] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [06/Dec/2019 10:12:35] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [06/Dec/2019 10:12:35] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [06/Dec/2019 10:12:35] "POST / HTTP/1.1" 200 -
```

8) Continue typing command to test if it is still working:

$ read /A/1.txt 0 12

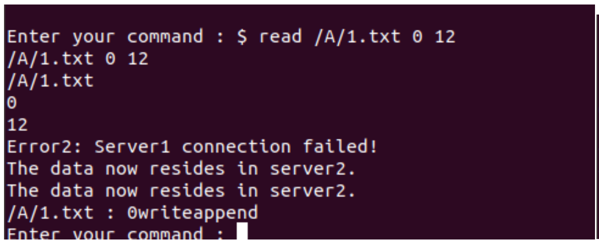
$ read /A/5.txt 0 12

$ create /A/6.txt

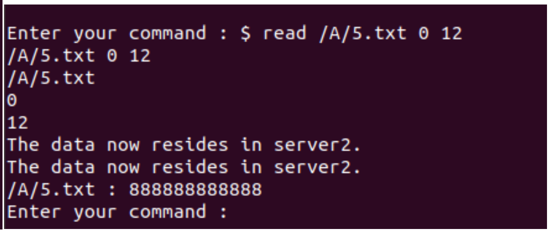
$ write /A/6.txt 99999999 0

$ create /A/7.txt

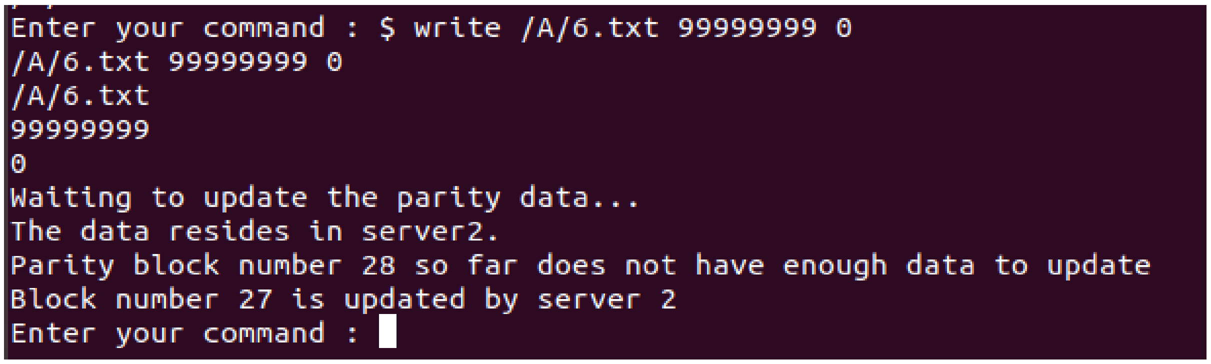
$write /A/7.txt 0000000000 0



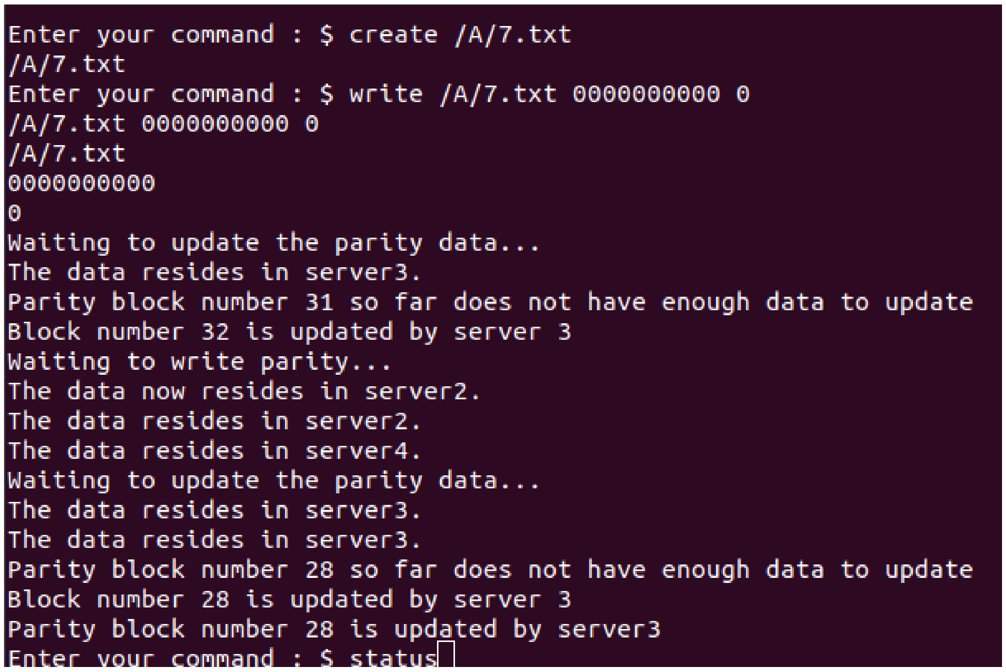
```
Enter your command : $ read /A/1.txt 0 12
/A/1.txt 0 12
/A/1.txt
0
12
Error2: Server1 connection failed!
The data now resides in server2.
The data now resides in server2.
/A/1.txt : 0writeappend
Enter your command :
```



```
Enter your command : $ read /A/5.txt 0 12
/A/5.txt 0 12
/A/5.txt
0
12
The data now resides in server2.
The data now resides in server2.
/A/5.txt : 888888888888
Enter your command :
```



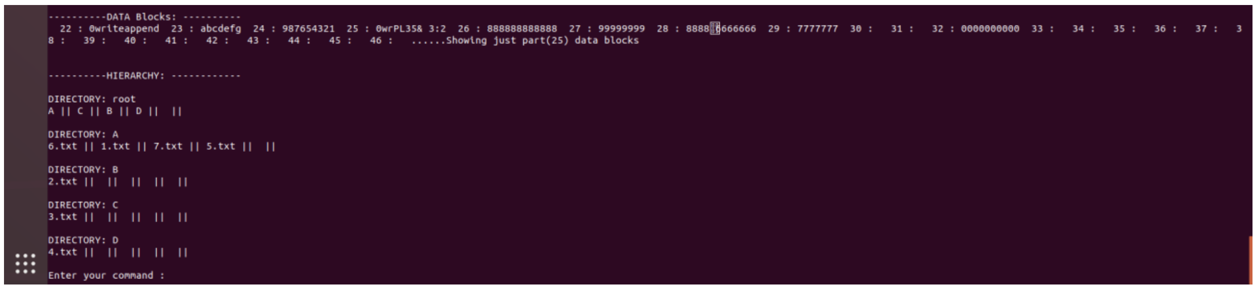
```
Enter your command : $ write /A/6.txt 99999999 0
/A/6.txt 99999999 0
/A/6.txt
99999999
0
Waiting to update the parity data...
The data resides in server2.
Parity block number 28 so far does not have enough data to update
Block number 27 is updated by server 2
Enter your command :
```



```
Enter your command : $ create /A/7.txt
/A/7.txt
Enter your command : $ write /A/7.txt 0000000000 0
/A/7.txt 0000000000 0
/A/7.txt
0000000000
0
Waiting to update the parity data...
The data resides in server3.
Parity block number 31 so far does not have enough data to update
Block number 32 is updated by server 3
Waiting to write parity...
The data now resides in server2.
The data resides in server2.
The data resides in server4.
Waiting to update the parity data...
The data resides in server3.
The data resides in server3.
Parity block number 28 so far does not have enough data to update
Block number 28 is updated by server 3
Parity block number 28 is updated by server3
Enter your command : $ status
```

9) The function can still work and another parity block is updated

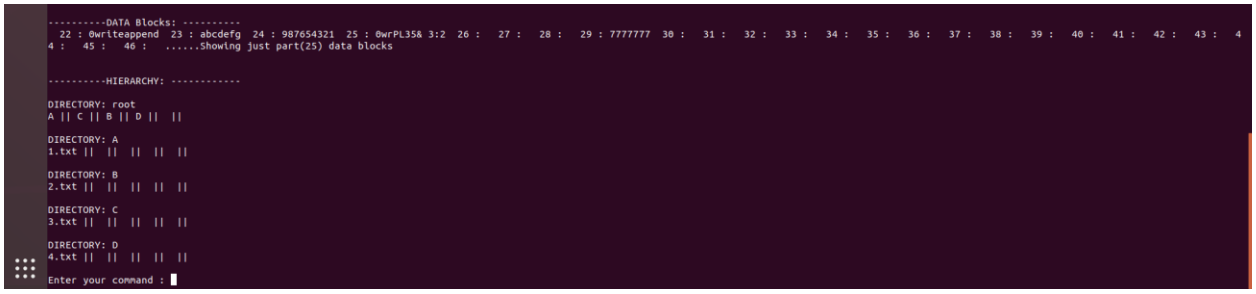
$ status



```
----------DATA Blocks: ----------
  22 : 0writeappend  23 : abcdefg  24 : 987654321  25 : 0wrPL35& 3:2  26 : 888888888888  27 : 99999999  28 : 88886666666  29 : 7777777  30 :   31 :   32 : 0000000000  33 :   34 :   35 :   36 :   37 :   3
8 :   39 :   40 :   41 :   42 :   43 :   44 :   45 :   46 :   ......Showing just part(25) data blocks


----------HIERARCHY: ------------

DIRECTORY: root
A || C || B || D ||  ||

DIRECTORY: A
6.txt || 1.txt || 7.txt || 5.txt ||  ||

DIRECTORY: B
2.txt ||  ||  ||  ||  ||

DIRECTORY: C
3.txt ||  ||  ||  ||  ||

DIRECTORY: D
4.txt ||  ||  ||  ||  ||

Enter your command :
```

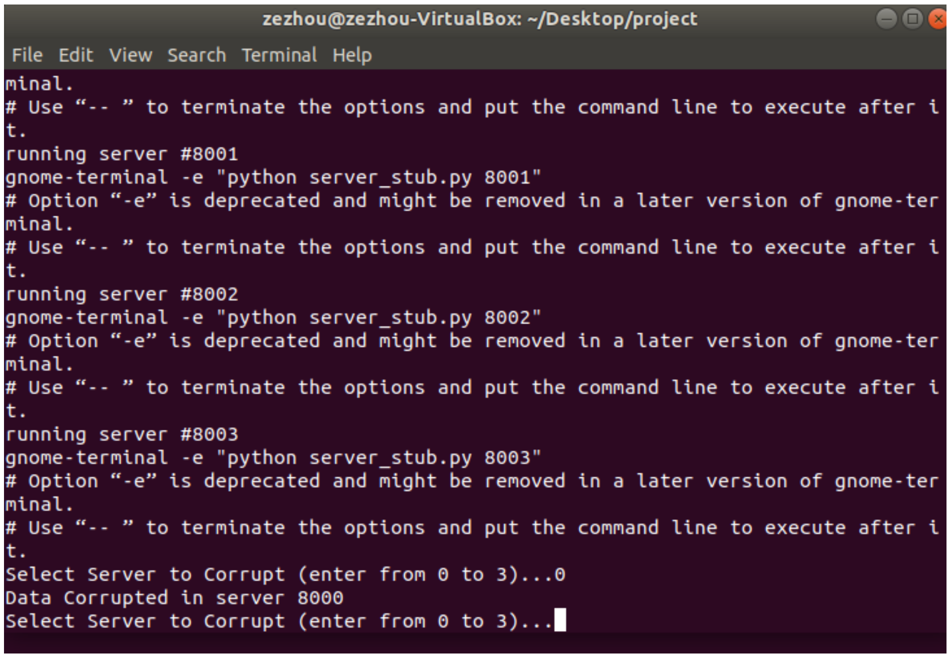
<u>For testing the RAID-5 function with corrupted data:</u>

1) CLOSE all terminals and restart
2) Still use the initial setup for testing



```
----------DATA Blocks: ----------
  22 : 0writeappend  23 : abcdefg  24 : 987654321  25 : 0wrPL35& 3:2  26 :   27 :   28 :   29 : 7777777  30 :   31 :   32 :   33 :   34 :   35 :   36 :   37 :   38 :   39 :   40 :   41 :   42 :   43 :   4
4 :   45 :   46 :   ......Showing just part(25) data blocks


----------HIERARCHY: ------------

DIRECTORY: root
A || C || B || D ||  ||

DIRECTORY: A
1.txt ||  ||  ||  ||  ||

DIRECTORY: B
2.txt ||  ||  ||  ||  ||

DIRECTORY: C
3.txt ||  ||  ||  ||  ||

DIRECTORY: D
4.txt ||  ||  ||  ||  ||

Enter your command :
```

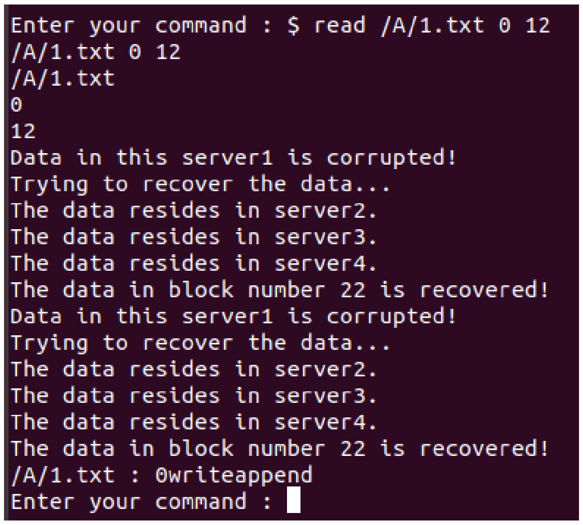
3) Select Server to Corrupt (type from 0 to 3) (This will make get data block return error)



```
zezhou@zezhou-VirtualBox: ~/Desktop/project
File  Edit  View  Search  Terminal  Help
minal.
# Use "-- " to terminate the options and put the command line to execute after i
t.
running server #8001
gnome-terminal -e "python server_stub.py 8001"
# Option "-e" is deprecated and might be removed in a later version of gnome-ter
minal.
# Use "-- " to terminate the options and put the command line to execute after i
t.
running server #8002
gnome-terminal -e "python server_stub.py 8002"
# Option "-e" is deprecated and might be removed in a later version of gnome-ter
minal.
# Use "-- " to terminate the options and put the command line to execute after i
t.
running server #8003
gnome-terminal -e "python server_stub.py 8003"
# Option "-e" is deprecated and might be removed in a later version of gnome-ter
minal.
# Use "-- " to terminate the options and put the command line to execute after i
t.
Select Server to Corrupt (enter from 0 to 3)...0
Data Corrupted in server 8000
Select Server to Corrupt (enter from 0 to 3)...
```

4) Try to read the corrupted data block from the server you choose:

$ read /A/1.txt 0 12



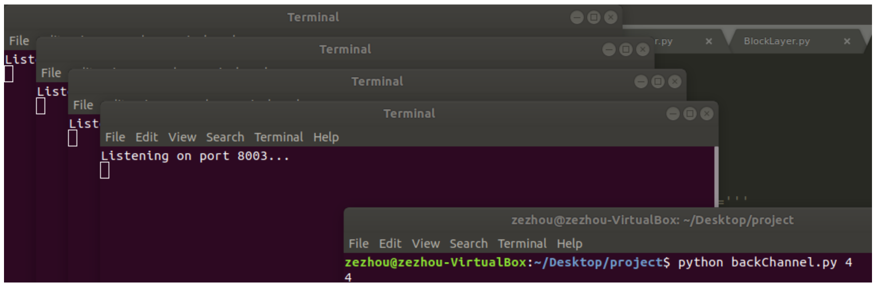
```
Enter your command : $ read /A/1.txt 0 12
/A/1.txt 0 12
/A/1.txt
0
12
Data in this server1 is corrupted!
Trying to recover the data...
The data resides in server2.
The data resides in server3.
The data resides in server4.
The data in block number 22 is recovered!
Data in this server1 is corrupted!
Trying to recover the data...
The data resides in server2.
The data resides in server3.
The data resides in server4.
The data in block number 22 is recovered!
/A/1.txt : 0writeappend
Enter your command :
```

<u>For testing the RAID-1 function with 1 server fail-stop:</u>

1) Open a terminal in Linux environment where all the python files are, and type:

python backChannel.py 4

where 4 means 4 servers, which can open 4 server ports: 8000, 8001, 8002, 8003, which can still work for RAID1

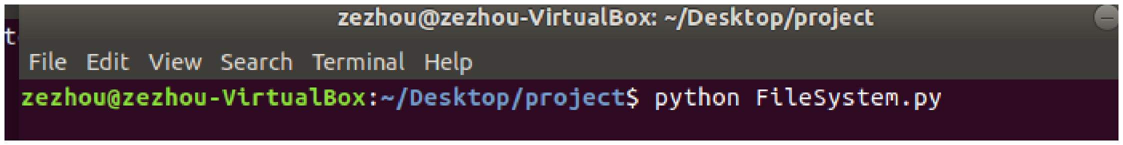


```
Listening on port 8003...
```

```
zezhou@zezhou-VirtualBox:~/Desktop/project$ python backChannel.py 4
4
```

2) Open another terminal also where the python files are, and type:

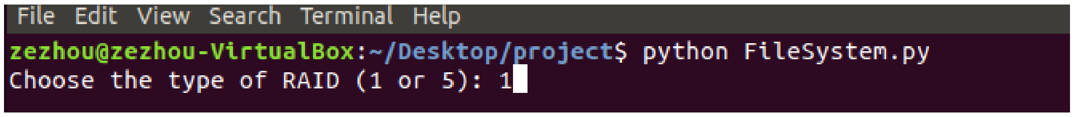
python FileSystem.py



zezhou@zezhou-VirtualBox: ~/Desktop/project
File Edit View Search Terminal Help

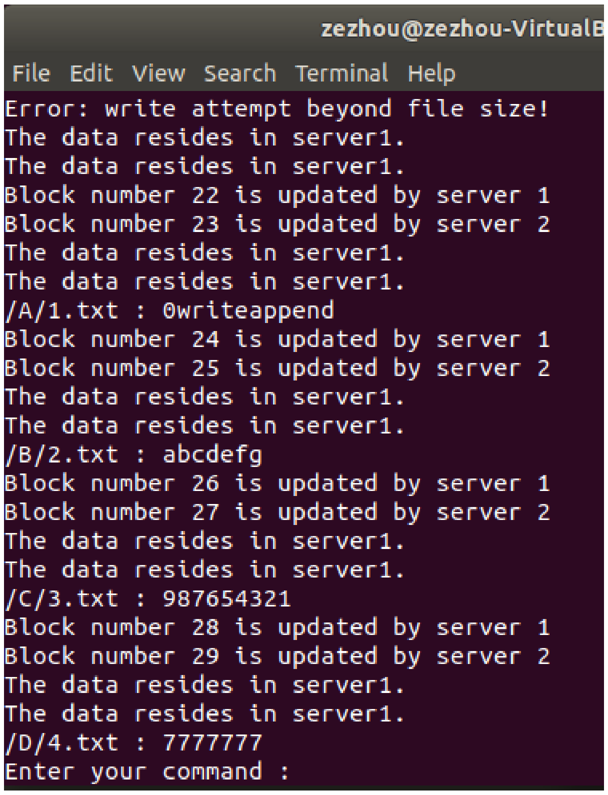
```
zezhou@zezhou-VirtualBox:~/Desktop/project$ python FileSystem.py
```

3) Choose which type of RAID you want to use (eg. Type ‘1’ for RAID1)



```
zezhou@zezhou-VirtualBox:~/Desktop/project$ python FileSystem.py
Choose the type of RAID (1 or 5): 1
```

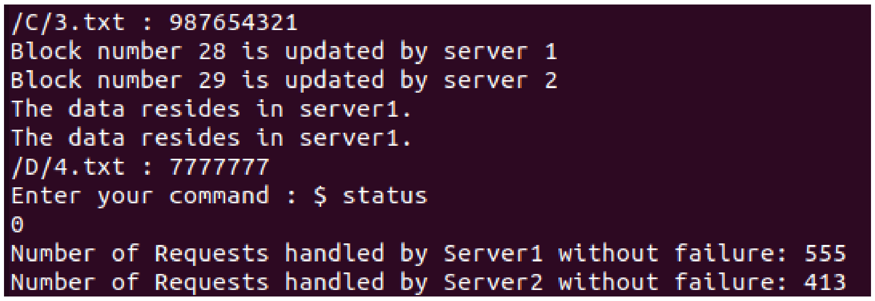
4) For ease the test process, use all the commands already in FileSystem.py:



```
Error: write attempt beyond file size!
The data resides in server1.
The data resides in server1.
Block number 22 is updated by server 1
Block number 23 is updated by server 2
The data resides in server1.
The data resides in server1.
/A/1.txt : 0writeappend
Block number 24 is updated by server 1
Block number 25 is updated by server 2
The data resides in server1.
The data resides in server1.
/B/2.txt : abcdefg
Block number 26 is updated by server 1
Block number 27 is updated by server 2
The data resides in server1.
The data resides in server1.
/C/3.txt : 987654321
Block number 28 is updated by server 1
Block number 29 is updated by server 2
The data resides in server1.
The data resides in server1.
/D/4.txt : 7777777
Enter your command :
```

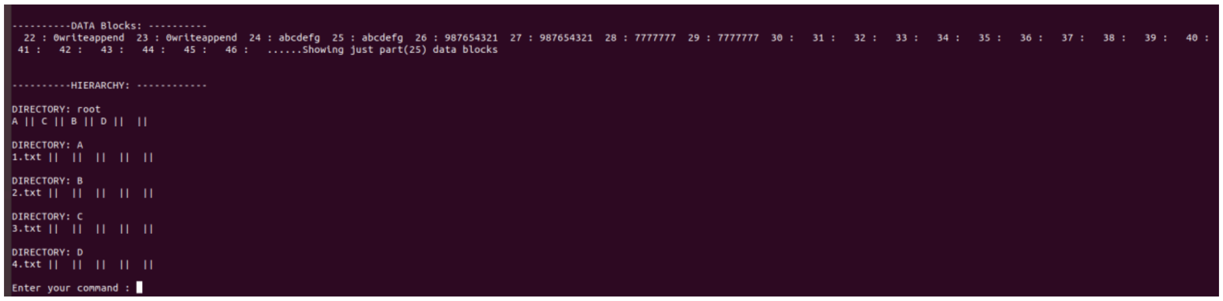
5) See the status of filesystem and check if all blocks and parity blocks lay out like RAID1

$ status



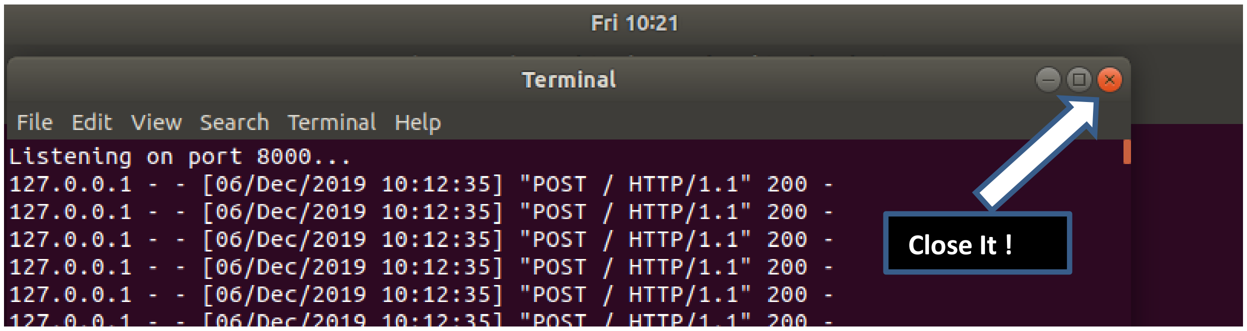
```
/C/3.txt : 987654321
Block number 28 is updated by server 1
Block number 29 is updated by server 2
The data resides in server1.
The data resides in server1.
/D/4.txt : 7777777
Enter your command : $ status
0
Number of Requests handled by Server1 without failure: 555
Number of Requests handled by Server2 without failure: 413
```

:



```
----------DATA Blocks: ----------
  22 : 0writeappend  23 : 0writeappend  24 : abcdefg  25 : abcdefg  26 : 987654321  27 : 987654321  28 : 7777777  29 : 7777777  30 :   31 :   32 :   33 :   34 :   35 :   36 :   37 :   38 :   39 :   40 :
 41 :   42 :   43 :   44 :   45 :   46 :   ......Showing just part(25) data blocks


----------HIERARCHY: -----------

DIRECTORY: root
A || C || B || D ||  ||

DIRECTORY: A
1.txt ||  ||  ||  ||  ||

DIRECTORY: B
2.txt ||  ||  ||  ||  ||

DIRECTORY: C
3.txt ||  ||  ||  ||  ||

DIRECTORY: D
4.txt ||  ||  ||  ||  ||

Enter your command :
```

6) Simply close one terminal to emulate 1 server fail-stop (eg. Close port 8000)

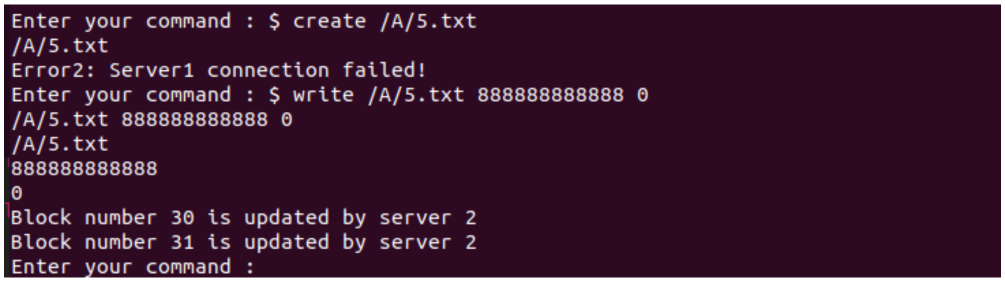


```
Listening on port 8000...
127.0.0.1 - - [06/Dec/2019 10:12:35] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [06/Dec/2019 10:12:35] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [06/Dec/2019 10:12:35] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [06/Dec/2019 10:12:35] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [06/Dec/2019 10:12:35] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [06/Dec/2019 10:12:35] "POST / HTTP/1.1" 200 -
```

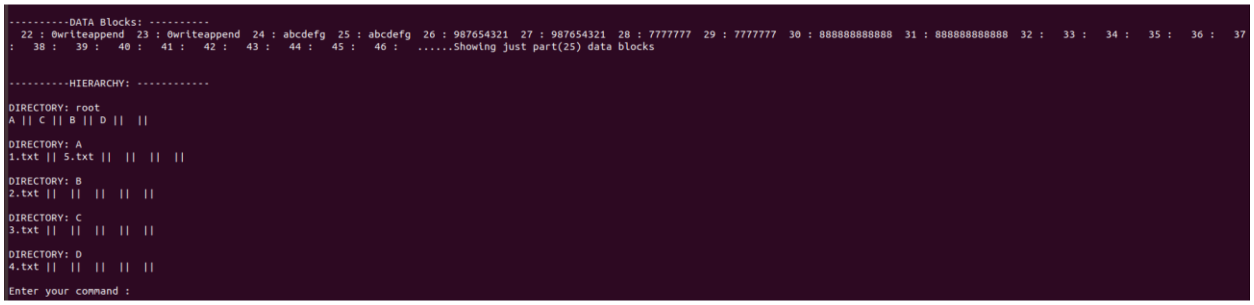
7) Continue typing command to test if it is still working:

$ create /A/5.txt

$ write /A/5.txt 888888888888 0



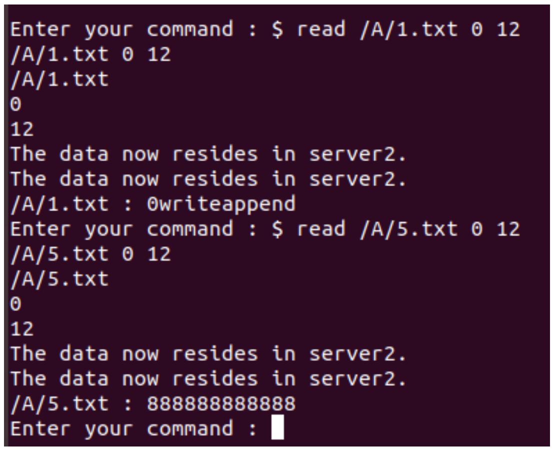
```
Enter your command : $ create /A/5.txt
/A/5.txt
Error2: Server1 connection failed!
Enter your command : $ write /A/5.txt 888888888888 0
/A/5.txt 888888888888 0
/A/5.txt
888888888888
0
Block number 30 is updated by server 2
Block number 31 is updated by server 2
Enter your command :
```



```
----------DATA Blocks: ----------
  22 : 0writeappend  23 : 0writeappend  24 : abcdefg  25 : abcdefg  26 : 987654321  27 : 987654321  28 : 7777777  29 : 7777777  30 : 888888888888  31 : 888888888888  32 :   33 :   34 :   35 :   36 :   37
:   38 :   39 :   40 :   41 :   42 :   43 :   44 :   45 :   46 :   ......Showing just part(25) data blocks


----------HIERARCHY: -----------

DIRECTORY: root
A || C || B || D ||  ||

DIRECTORY: A
1.txt || 5.txt ||  ||  ||  ||

DIRECTORY: B
2.txt ||  ||  ||  ||  ||

DIRECTORY: C
3.txt ||  ||  ||  ||  ||

DIRECTORY: D
4.txt ||  ||  ||  ||  ||

Enter your command :
```

$ read /A/1.txt 0 12

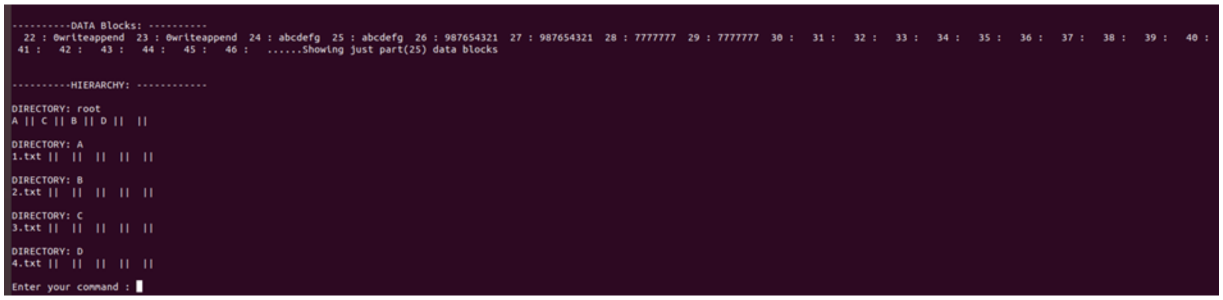
$ read /A/5.txt 0 12



```
Enter your command : $ read /A/1.txt 0 12
/A/1.txt 0 12
/A/1.txt
0
12
The data now resides in server2.
The data now resides in server2.
/A/1.txt : 0writeappend
Enter your command : $ read /A/5.txt 0 12
/A/5.txt 0 12
/A/5.txt
0
12
The data now resides in server2.
The data now resides in server2.
/A/5.txt : 888888888888
Enter your command :
```

For testing the RAID-1 function with corrupted data:

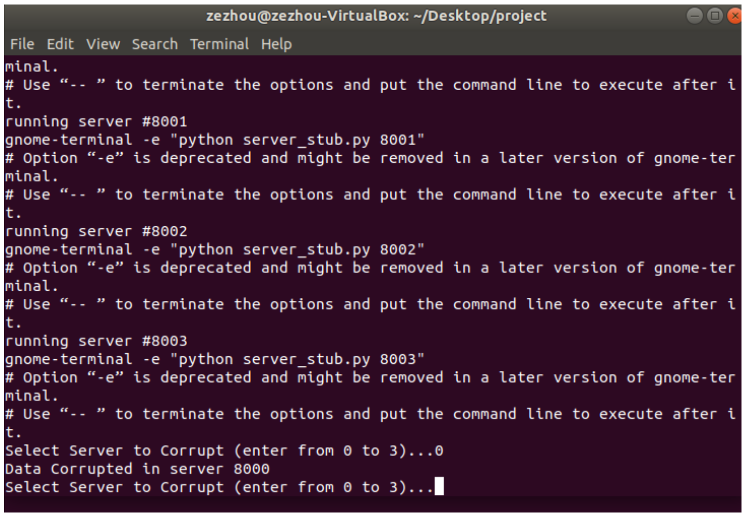
1) still using the initial setup for testing



```
----------DATA Blocks: ----------
 22 : 0writeappend  23 : 0writeappend  24 : abcdefg  25 : abcdefg  26 : 987654321  27 : 987654321  28 : 7777777  29 : 7777777  30 :   31 :   32 :   33 :   34 :   35 :   36 :   37 :   38 :   39 :   40 :
 41 :   42 :   43 :   44 :   45 :   46 :   ......Showing just part(25) data blocks


----------HIERARCHY: ------------

DIRECTORY: root
A || C || B || D ||  ||

DIRECTORY: A
1.txt ||  ||  ||  ||  ||

DIRECTORY: B
2.txt ||  ||  ||  ||  ||

DIRECTORY: C
3.txt ||  ||  ||  ||  ||

DIRECTORY: D
4.txt ||  ||  ||  ||  ||

Enter your command : █
```

22: 0writeappend	23: 0writeappend
24: abcdefg	25: abcdefg
26: 987654321	27: 987654321
28: 7777777	29: 7777777

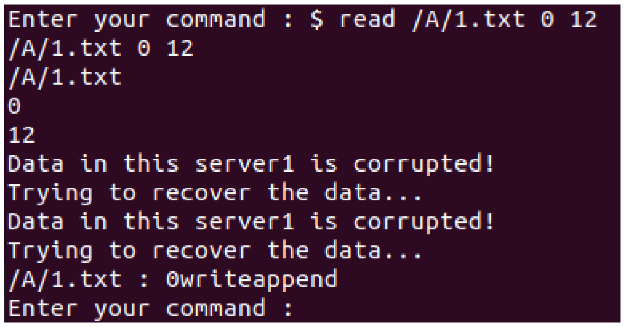
2) Select Server to Corrupt (type from 0 to 3) (This will make get data block return error)


zezhou@zezhou-VirtualBox: ~/Desktop/project
File Edit View Search Terminal Help

```
minal.
# Use “-- ” to terminate the options and put the command line to execute after i
t.
running server #8001
gnome-terminal -e "python server_stub.py 8001"
# Option “-e” is deprecated and might be removed in a later version of gnome-ter
minal.
# Use “-- ” to terminate the options and put the command line to execute after i
t.
running server #8002
gnome-terminal -e "python server_stub.py 8002"
# Option “-e” is deprecated and might be removed in a later version of gnome-ter
minal.
# Use “-- ” to terminate the options and put the command line to execute after i
t.
running server #8003
gnome-terminal -e "python server_stub.py 8003"
# Option “-e” is deprecated and might be removed in a later version of gnome-ter
minal.
# Use “-- ” to terminate the options and put the command line to execute after i
t.
Select Server to Corrupt (enter from 0 to 3)...0
Data Corrupted in server 8000
Select Server to Corrupt (enter from 0 to 3)...█
```

3) Try to read the corrupted data block from the server you choose:

$ read /A/1.txt 0 12



```
Enter your command : $ read /A/1.txt 0 12
/A/1.txt 0 12
/A/1.txt
0
12
Data in this server1 is corrupted!
Trying to recover the data...
Data in this server1 is corrupted!
Trying to recover the data...
/A/1.txt : 0writeappend
Enter your command :
```

Performance Results

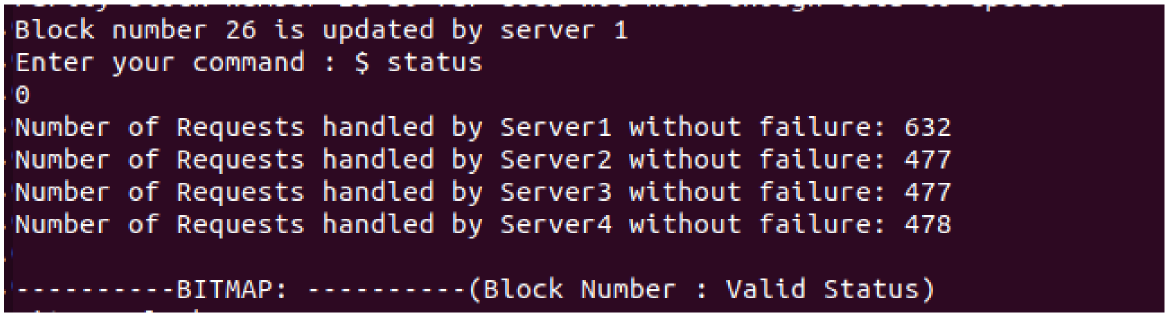
The performance results can be seen in command: *$ status*

The test cases are still the initial start part.

RAID5:

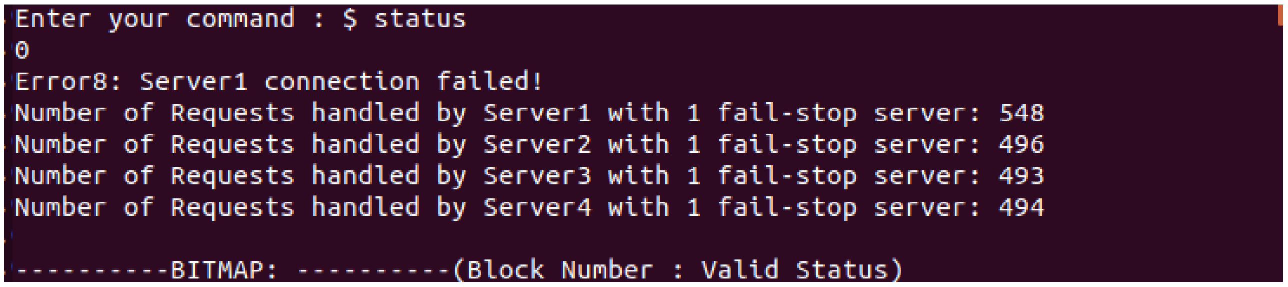
For the Initial start, the requests server1 handled are 548:

Without server1 fail-stop, one server will be 632 requests.



```
Block number 26 is updated by server 1
Enter your command : $ status
0
Number of Requests handled by Server1 without failure: 632
Number of Requests handled by Server2 without failure: 477
Number of Requests handled by Server3 without failure: 477
Number of Requests handled by Server4 without failure: 478

----------BITMAP: ----------(Block Number : Valid Status)
```

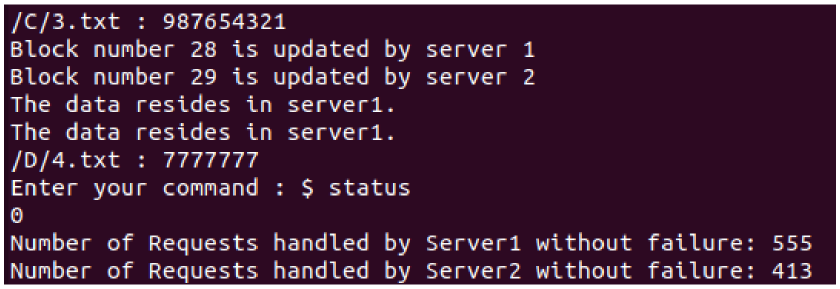
With server1 fail-stop,



```
Enter your command : $ status
0
Error8: Server1 connection failed!
Number of Requests handled by Server1 with 1 fail-stop server: 548
Number of Requests handled by Server2 with 1 fail-stop server: 496
Number of Requests handled by Server3 with 1 fail-stop server: 493
Number of Requests handled by Server4 with 1 fail-stop server: 494

----------BITMAP: ----------(Block Number : Valid Status)
```

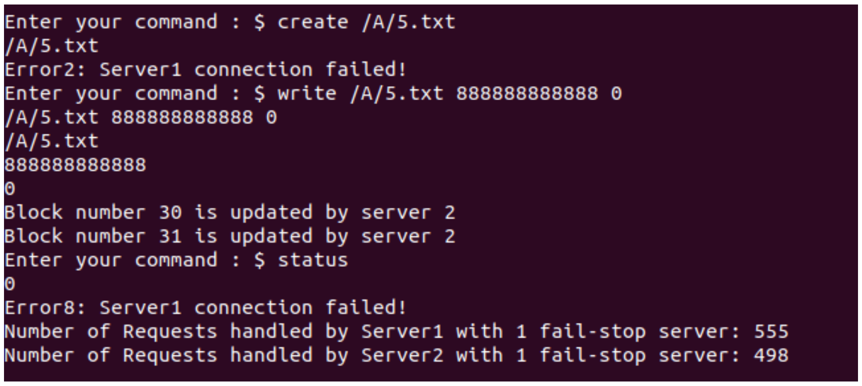
RAID1:

For the Initial start, the requests server1 handled are 555:



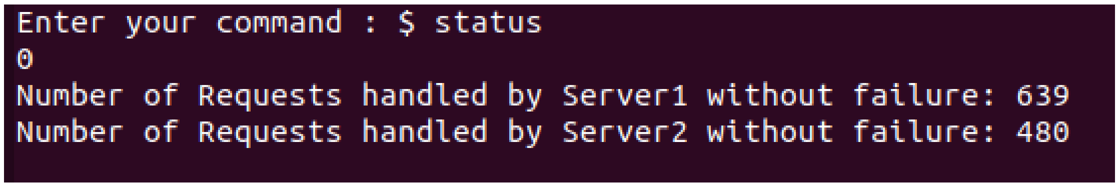
```
/C/3.txt : 987654321
Block number 28 is updated by server 1
Block number 29 is updated by server 2
The data resides in server1.
The data resides in server1.
/D/4.txt : 7777777
Enter your command : $ status
0
Number of Requests handled by Server1 without failure: 555
Number of Requests handled by Server2 without failure: 413
```

With server1 fail-stop,



```
Enter your command : $ create /A/5.txt
/A/5.txt
Error2: Server1 connection failed!
Enter your command : $ write /A/5.txt 888888888888 0
/A/5.txt 888888888888 0
/A/5.txt
888888888888
0
Block number 30 is updated by server 2
Block number 31 is updated by server 2
Enter your command : $ status
0
Error8: Server1 connection failed!
Number of Requests handled by Server1 with 1 fail-stop server: 555
Number of Requests handled by Server2 with 1 fail-stop server: 498
```

Without server1 fail-stop,



```
Enter your command : $ status
0
Number of Requests handled by Server1 without failure: 639
Number of Requests handled by Server2 without failure: 480
```