DISTRIBUTED FILE SYSTEM

Shreenivas Pai N -78236469 Harshal Patil -55528581

Graduate Graduate

Problem Statement

This project implements a distributed file system where in there is a client/server based architecture. It includes single robust remote meta server to store all the meta data of the file system and multiple data servers which stores the data using the concepts of distributed file system like redundancy and fault tolerance. Redundancy reduces the load by equally distributing the data among the other servers. In this project 2 replicas along with one original copy is stored on multiple data servers in round-robin fashion, which ensures that, if n-2 servers are alive the file system would still be able to retrieve the data correctly in proper order. In order to maintain the correctness and validity of the data an error correction method using checksum is implemented. Whenever the data is read, the file system 1st validates the checksum, if the checksum matches the data is fetched correctly. If the checksum fails to match the data is to be retrieved from one of the redundant replica which is stored in the next server in round-robin fashion. In case, if the 1st replica fails to match the checksum, 2nd replica is used to fetch the data along with the validation using checksum. Additionally the advantage of redundant distributed file system is that even if the server fails, client doesn't have to wait until the failed server becomes active again to read the data. It still can fetch the data from the replicas stored on the other servers. The file system also implements a persistent storage wherein we create a file of all the data for every data server and if a server fails it can recover the complete state as of before the crash from its persistent storage file. The main database is in the disk and the file system always writes to the disk first before returning an RPC call. For implementing the persistent storage the python Shelve has benn used. Even when the persistent file fails and the server is crashed the file system can still retrieve the data from the replicas and create the persistent file and restore its state.

Design

This project implements a distributed file system where in there is a client/server based architecture. It includes single robust remote meta server to store all the meta data of the file system and multiple data servers which stores the data using the concepts of distributed file system like redundancy and fault tolerance. Redundancy reduces the load by equally distributing the data among the other servers. In this project 2 replicas along with one original copy is stored on multiple data servers in round-robin fashion, which ensures that, if n-2 servers are alive the file system would still be able to retrieve the data correctly in proper order. In order to maintain the correctness and validity checksum is used by calculating its hash value.

This project consists of 3 main files

1. Metaserver.py
2. Dataserver.py
3. DistributedFS.py

Meta server consists of metadata of all files and directories. In this project we consider the meta server to robust and it never fails. It consists of two dicts. One for storing the meta files and another for storing the parent and contents of the directory.

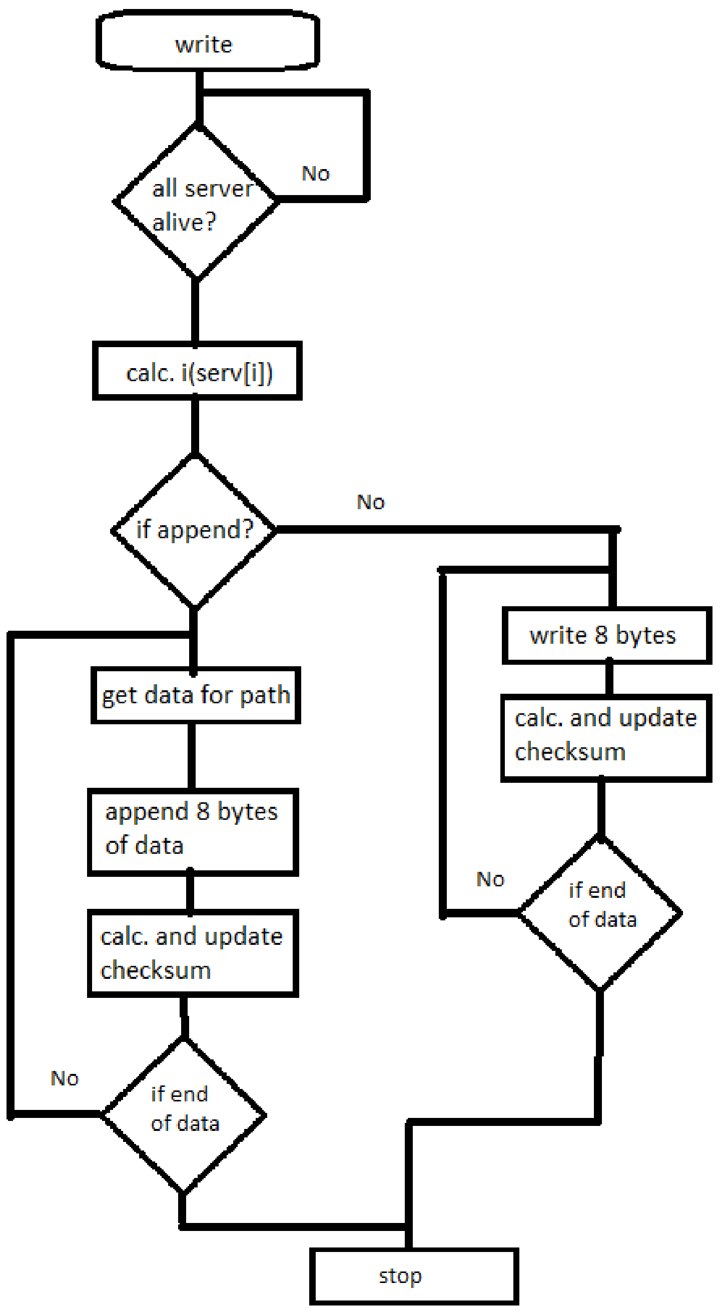
Dataserver are multiple and project can support up to 5 dataservers assuming dataservers are not robust we created the redundant file system which keeps replicas of the original data block and it is stored in the next adjacent servers in round robin fashion. The dataserver consists of 3 dicts to store original, replica1 and replica2. Also, to maintain the validity of the data checksum is used. 3 dicts are used for storing checksum original, replica1 and replica 2. Each dataserver has persistent storage which stores all the data of corresponding server in disk. Whenever the dataserver fails, the data is retrieved from this storage. There is a chance that the persistence storage fails and that can be recovered from the redundant copies, and this will be done by the client. Shelve function is used to create the "data store".

DistributedFS.py:

The client will support most of the functionalities that is supported by the Fuse file system. And some of them are explained below.

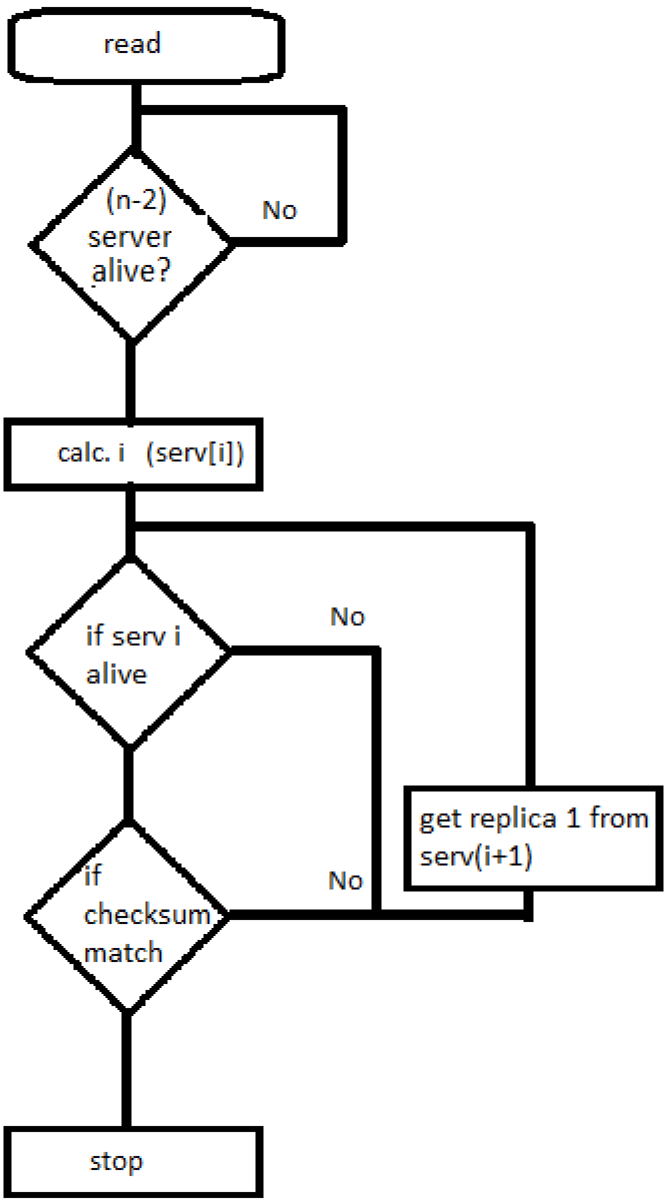
1. **Write**
   When a write function is called the client will first check the availability of the server and if all the servers are alive it will proceed to write else it will wait until all the servers are alive. To check this functionality a separate function is written. When all the servers are alive it will get the

data from the respective server which is calculated using hash function. This hash function always will return a different value for different path. Write is done 8 bytes at a time and stored as list in the server. Same set of data is written in replica 1 and replica 2 in their respective servers. Alongside checksum is calculated and updated in respective server. Append is also included in write functionality. Separate function is used to do truncate. Following flowchart will give a flow of write operation.



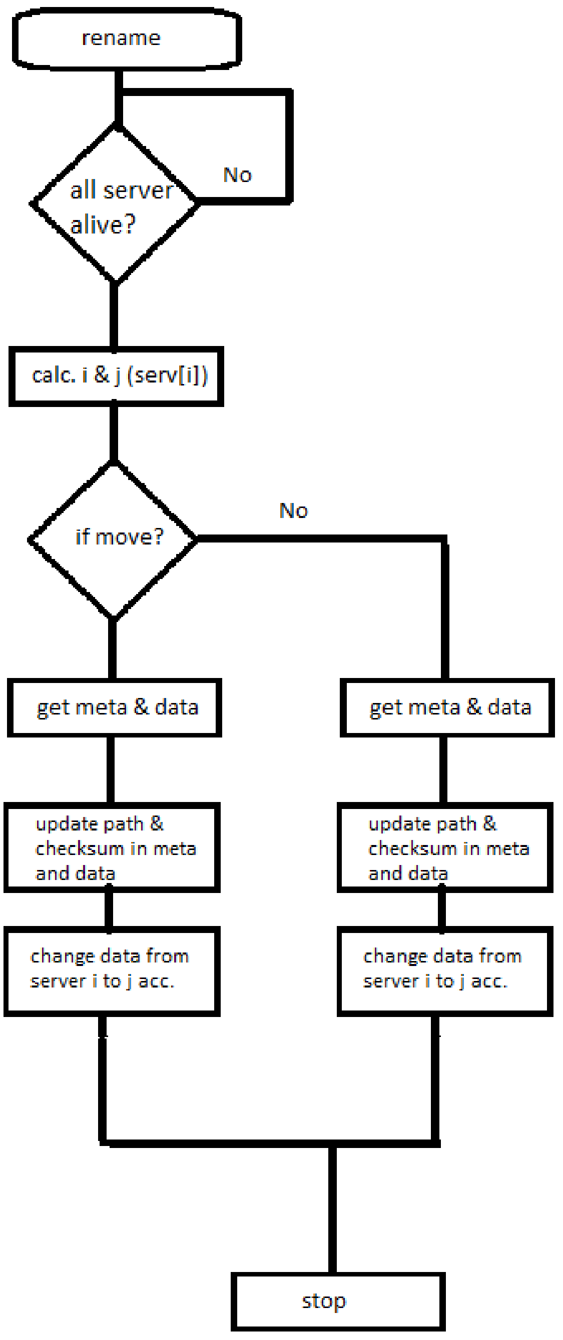
2. Read

When a read function is called for a path, it will first check the availability on n-2 servers and if available it will proceed. It will calculate the hash for path and decide the server to read. If the calculated server is alive it will check for checksum. If the checksum is validated it will return. If it fails to validate the checksum or if the server is not alive it will go for the replicas of it and do the same.



3. Rename

Rename will start with checking of servers and if all the servers are available it will proceed. The hash value for both the old and new path is calculated and corresponding servers are found to be replaced. If move or rename is called corresponding functionality is performed. It will update the path in metserver and dataserver. When the path is updated in dataserver the server position is also updated i.e it is placed in the exact position where the hash value is calculated for the newpath.


rename
all server alive?
No
calc. i & j (serv[i])
if move?
No
get meta & data
get meta & data
update path & checksum in meta and data
update path & checksum in meta and data
change data from server i to j acc.
change data from server i to j acc.
stop

4. Symlink
   symlink is also called soft link it is linked to the name of a file. When a symlink to a file is created anywhere the contents of the original files can be read properly. symlink can be moved from directory to directory without affecting the contents. symlink can also be deleted and it results in no changes to the original file.

Test cases passed

1.Create a text file -1.txt

2. Read the text file –cat 1.txt

3.copy the data from 1.txt to 2.txt

4.Read the copied file 2.txt

5. Append to the copied file

6. Read the file you just appended to

7.rename the file 2.txt to 3.txt

8.create a directory in root ‘a’

9. Go inside the directory and create a text file 4.txt

10.move the the file 3.txt to the directory ‘a’

11.ls in the directory ‘a’ is showing 4.txt and 3.txt

12.create a empty directory in root ‘b’

13.move the directory a to b

14.go inside directory b and perform ls it shows 4.txt and 3.txt

15.create another directory c in the root and create a file inside it.

16.move the directoy ‘b’ to ‘c’ it shows the contents of both b and also contain the directory b.

17.Rename the directory ‘c’ to ‘d’

18.delete an empty directory using rmdir it should get deleted .

19.delete a directory with contents using rmdir it should give an error.

20. delete a directory with contents using rm-rf should delete the directory.

21. create a symlink in the directory of a file that is in root.

22. check for the source of the symlink

23. move the symlink to any other directory.

24. create a directory called ‘st_mode’ and check its stats.

25. change the ownership of a directory

26.change the ownership of a file

27.change the permissions given to a file.

28. truncate the text file should reduce the length of the file.

29.truncate the text file to size that is larger than the file size the contents should remain the same.

30.truncate the file size to 0 it should not show any contents now.

31.crash any one server and try to read data it should be able to read the data.

32. crash any 2 servers and try to read data it should be able to read the data.

33.crash 3 servers and try to read it should wait until N-2 servers are alive.

34.crash a server and write data it should wait for all the servers to alive.

35.get all the server up and the write should automatically be completed.

36.delete a persistent storage file and crash the server and bring it back up again it should be able to read normally and will remake the data store file.

37.crash a server and delete the data store file and do the write operation it should wait for all the severs to be up.

38. get the servers back up again and the write operation should be completed and the data store should be created by itself.

Conclusion

The Distributed file system has been successfully implemented. The filesystem can successfully perform all the basic operations like read, write, truncate, append, create, move, rename and others. The file system can support N=4 as well as N=5 data servers. Successfully able to store the data of the file on multiple data servers in round robin fashion. The system has also implemented checksum for ensuring the validity of the data. Even if up to 2 servers fail and crash at the same time the client is able to read all the data stored on the data servers, hence redundancy has been implemented successfully. The file system also implements a persistent storage wherein if any server crashes and after a certain period comes back alive the server is able to retrieve its complete state before resuming operation from the persistent storage. The file system is also capable of handling a case where the persistent storage fails. The filesystem is successfully able to check if the data received in the file is corrupt even when there is one data corruption per server in the same path in non adjacent server as well as corruption of multiple unique paths in different data servers. Once the file system knows that the data is corrupted it updates the data servers with the correct values from the replicas. Whenever a one or two servers are down the file system can do the read operation but blocks all write operation in case of server failures.

Group effort

Both the members have contributed equally in the development of the project. Both the members discussed the problem statement and decided on the algorithm together and implemented the project in parts.

Code for client:

```
#!/usr/bin/env python


from __future__ import print_function, absolute_import, division

from fuse import FUSE, FuseOSError, Operations, LoggingMixIn

import logging, xmlrpclib, pickle

from xmlrpclib import Binary

import os, hashlib,socket, time


from collections import defaultdict

from errno import ENOENT, ENOTEMPTY

from stat import S_IFDIR, S_IFLNK, S_IFREG

from sys import argv, exit

from time import time

from time import sleep


if not hasattr(__builtins__, 'bytes'):

    bytes = str
```

```
def _get_rpc(sport):
    a = xmlrpclib.ServerProxy("http://localhost:"+str(int(sport)))

    try:
            a._()
    except xmlrpclib.Fault:
        # connected to the server
        pass
    except socket.error:
        # Not connected
        return False, None
                    # Just in case the method is registered in the XmlRPC server
    return True, a


def check_server():
    global serv, scount
    serv=[None]*scount
    global ports,scount
    connected=[0]*scount
    stats=[0]*scount
    while 1:
        i=0
        sum1=0
        while i<scount:
            connected[i],serv[i]= _get_rpc(ports[i])
            i+=1
        i=0
        while i<scount:
            if connected[i]==True:
                print ("server"+str(i)+" is alive")
                stats[i]=1
            if connected[i]!=True:
                print("Waiting for server"+str(i))
            i+=1
        for every in stats:
            sum1+=every
        if sum1==scount:
            print ("All servers are ready for contact")
                if xyz==1:
                        restore_per()
            break


xyz=0
def restore_per():
    global scount
    i=0
    while i<scount:
            check=checkf(serv[i], i)
            if not check:
                x=i
                x1=(x+1)%scount
                x2=(x-1)%scount
                dat=r1getdata3(serv[x1], x1)
                dat1=r2getdata3(serv[x1], x1)
                dat2=r1getdata3(serv[x2], x2)
                putdata3(serv[x], dat, x)
                r1putdata3(serv[x], dat1, x)
                r2putdata3(serv[x], dat2, x)
                dat=cr1getdata3(serv[x1], x1)
                dat1=cr2getdata3(serv[x1], x1)
                dat2=cr1getdata3(serv[x2], x2)
                cputdata3(serv[x], dat, x)
                cr1putdata3(serv[x], dat1, x)
                cr2putdata3(serv[x], dat2, x)
            i+=1
```

```
def check_server_read():
    global serv
    global ports,scount, xyz
    global serv_read
    global readcheckserv
    readcheckserv=[0]*scount
    connected=[None]*scount
    while 1:
        i=0
        sum1=0
        while i<scount:
            connected[i],serv[i]= _get_rpc(ports[i])
            i+=1
        i=0
        while i<scount:
            if connected[i]==True:
                print ("server"+str(i)+" is alive")
                readcheckserv[i]=1
                            check=checkf(serv[i], i)
                            if not check:
                                        restore_per()
            if connected[i]!=True:
                print("Waiting for server"+str(i))
            i+=1
        for every in readcheckserv:
            sum1+=every

        if sum1>=(scount-2):
            print ("atleast "+ str(sum1) +" servers are ready for contact")
            break
    return sum1
```

```
def checkf(server,x):
    return pickle.loads((serv[x].checkpr()).data)

def putdata1(server,key,value):
    return server.put(Binary(key),Binary(pickle.dumps(value)))

def getdata1(server,key):
    return pickle.loads((server.get(Binary(key))).data)



def putdata2(server,key,value,x):
    return serv[x].put(Binary(key),Binary(pickle.dumps(value)))

def getdata2(server,key,x):
    return pickle.loads((serv[x].get(Binary(key))).data)



def getdata3(server,x):
    return pickle.loads((serv[x].get3()).data)

def putdata3(server,value,x):
    return serv[x].put3(Binary(pickle.dumps(value)))

def r1putdata2(server,key,value,x):
    return serv[x].r1put(Binary(key),Binary(pickle.dumps(value)))

def r1getdata2(server,key,x):
    return pickle.loads((serv[x].r1get(Binary(key))).data)

def r1getdata3(server,x):
    return pickle.loads((serv[x].r1get3()).data)

def r1putdata3(server,value,x):
    return serv[x].r1put3(Binary(pickle.dumps(value)))
```

```
def r2putdata2(server,key,value,x):
    return serv[x].r2put(Binary(key),Binary(pickle.dumps(value)))

def r2getdata2(server,key,x):
    return pickle.loads((serv[x].r2get(Binary(key))).data)

def r2getdata3(server,x):
    return pickle.loads((serv[x].r2get3()).data)

def r2putdata3(server,value,x):
    return serv[x].r2put3(Binary(pickle.dumps(value)))

def cputdata2(server,key,value,x):
    return serv[x].cput(Binary(key),Binary(pickle.dumps(value)))

def cgetdata2(server,key,x):
    return pickle.loads((serv[x].cget(Binary(key))).data)

def cgetdata3(server,x):
    return pickle.loads((serv[x].cget3()).data)

def cputdata3(server,value,x):
    return serv[x].cput3(Binary(pickle.dumps(value)))

def cr1putdata2(server,key,value,x):
    return serv[x].r1cput(Binary(key),Binary(pickle.dumps(value)))

def cr1getdata2(server,key,x):
    return pickle.loads((serv[x].r1cget(Binary(key))).data)

def cr1getdata3(server,x):
    return pickle.loads((serv[x].r1cget3()).data)
```

```
def cr1putdata3(server,value,x):
    return serv[x].r1cput3(Binary(pickle.dumps(value)))


def cr2putdata2(server,key,value,x):
    return serv[x].r2cput(Binary(key),Binary(pickle.dumps(value)))


def cr2getdata2(server,key,x):
    return pickle.loads((serv[x].r2cget(Binary(key))).data)


def cr2getdata3(server,x):
    return pickle.loads((serv[x].r2cget3()).data)


def cr2putdata3(server,value,x):
    return serv[x].r2cput3(Binary(pickle.dumps(value)))



def corruptf(server,path,x):
    return serv[x].corrupt(Binary(pickle.dumps(path)))



class Memory(LoggingMixIn, Operations):


    def stringtolist(self,s):
        L=[]
        while s!="":
            L.append(s[0:8])
            s=s[8:]
        return L



    def listtostring(self,l):
        s=''.join(l)
        return s
```

```
def __init__(self):
    self.primary = {}
    self.primary['files'] = {}
          self.primary['child'] = defaultdict(list)
          self.primary['pos'] = defaultdict(list)
    self.fd = 0
    now = time()
    self.primary['files']['/'] = dict(st_mode=(S_IFDIR | 0o755), st_ctime=now,
                        st_mtime=now, st_atime=now, st_nlink=2)
    putdata1(server,"files",self.primary['files'])
    putdata1(server,"child",self.primary['child'])

def chmod(self, path, mode):
    x = getdata1(server,"files")
    x[path]['st_mode'] &= 0o770000
    x[path]['st_mode'] |= mode
    putdata1(server,"files",x)
    return 0



def chown(self, path, uid, gid):
    x = getdata1(server,"files")
    x[path]['st_uid'] = uid
    x[path]['st_gid'] = gid
    putdata1(server,"files",x)

def create(self, path, mode):
          global serv, scount, xyz
    check_server()
          xyz=1
    fl1 = getdata1(server,"files")
    cl1 = getdata1(server,"child")
    fl1[path] = dict(st_mode=(S_IFREG | mode), st_nlink=1,
                         st_size=0, st_ctime=time(), st_mtime=time(),
                        st_atime=time())
         i=0
         while(i<scount):
                putdata2(serv[i], path , [],i)
                r1putdata2(serv[i], path , [],i)
                r2putdata2(serv[i], path , [],i)
                cputdata2(serv[i], path , [],i)
                cr1putdata2(serv[i], path , [],i)
                cr2putdata2(serv[i], path , [],i)
                      i+=1
    parentpath=os.path.dirname(path)
         childpath=os.path.basename(path)
    cl1[parentpath].append(childpath)
    self.fd += 1
         putdata1(server,"files",fl1)
         putdata1(server,"child",cl1)
    return self.fd

def getattr(self, path, fh=None):
    fl1 = getdata1(server,"files")
    if path not in fl1:
        raise FuseOSError(ENOENT)
    return fl1[path]

def getxattr(self, path, name, position=0):
    fl1 = getdata1(server,"files")
    attrs = fl1[path].get('attrs', {})
    try:
        return attrs[name]
    except KeyError:
        return ''

def listxattr(self, path):
    fl1 = getdata1(server,"files")
    attrs = fl1[path].get('attrs', {})
    return attrs.keys()

def mkdir(self, path, mode):
    fl1 = getdata1(server,"files")
    cl1 = getdata1(server,"child")
    fl1[path] = dict(st_mode=(S_IFDIR | mode), st_nlink=2,
                            st_size=0, st_ctime=time(), st_mtime=time(),
                            st_atime=time())

    parentpath=os.path.dirname(path)
          childpath=os.path.basename(path)
    cl1[parentpath].append(childpath)
    cl1[path]=[]
    fl1[parentpath]['st_nlink'] += 1
    putdata1(server,"files",fl1)
    putdata1(server,"child",cl1)



def open(self, path, flags):
    self.fd += 1
    return self.fd



def read(self, path, size, offset, fh):
          global readcheckserv, scount
    sum1=check_server_read()
          if sum1>=(scount-2):
              x=hash(path)%scount
              x1=(x+1)%scount
              x2=(x+2)%scount
        a1=path
        fl1 = getdata1(server,"files")
        leng= (fl1[path]['st_size'])
```

```
if leng%8 == 0:
  leng=int(leng/8)
else:
    leng=int((leng/8)+1)
element=0
element_cond=0
dat=[]
data2=[]
while leng!=0:
    flag=0
    flag1=0
    flag2=0
    flag3=0
    if readcheckserv[x] == 1 and flag==0:
          checksum=cgetdata2(serv[x],path,x)
          echecksum=checksum[element]
          tocheck=getdata2(serv[x], path,x)
          etocheck=tocheck[element]
   if echecksum == hash(etocheck):
                data2=etocheck
                flag=1
                flag1=1
    if readcheckserv[x1] == 1 and flag==0:
          r1checksum=cr1getdata2(serv[x1],path,x1)
          er1checksum=r1checksum[element]
          r1tocheck=r1getdata2(serv[x1], path,x1)
   er1tocheck=r1tocheck[element]
          if er1checksum == hash(er1tocheck):
                data2=er1tocheck
                flag=1
                flag2=1
    if readcheckserv[x2] == 1 and flag==0:
          r2checksum=cr2getdata2(serv[x2],path,x2)
          er2checksum=r2checksum[element]
```

```
            r2tocheck=r2getdata2(serv[x2], path,x2)
            er2tocheck=r2tocheck[element]
            if er2checksum == hash(er2tocheck):
                data2=er2tocheck
                flag=1
                flag3=1

    if readcheckserv[x] == 1 and readcheckserv[x1] == 1:
            if flag1==0 and flag2==0 and flag3==1:
     tocheck[element]=data2
              checksum[element]=hash(data2)
     r1tocheck[element]=data2
              r1checksum[element]=hash(data2)
                cputdata2(serv[x],path,checksum,x)
                cr1putdata2(serv[x1],path,r1checksum,x1)
                putdata2(serv[x],path,data2,x)
                r1putdata2(serv[x1],path,data2,x1)
            if flag1==0 and flag2==1:
     tocheck[element]=data2
              checksum[element]=hash(data2)
                cputdata2(serv[x],path,checksum,x)
                putdata2(serv[x],path,tocheck,x)
    if readcheckserv[x] == 1 and readcheckserv[x1] == 0:
            if flag1==0 and flag3==1:
                tocheck[element]=data2
              checksum[element]=hash(data2)
                cputdata2(serv[x],path,checksum,x)
                putdata2(serv[x],path,tocheck,x)
    if readcheckserv[x] == 0 and readcheckserv[x1] == 1:
            if flag2==0 and flag3==1:
     r1tocheck[element]=data2
              r1checksum[element]=hash(data2)
                cr1putdata2(serv[x1],path,r1checksum,x1)
                r1putdata2(serv[x1],path,r1tocheck,x1)
```

```
                         dat.append(data2)
                         element_cond=(element_cond+1)%scount
                         if element_cond == 0:
                                       element+=1
                  x=(x+1)%scount
                         x1=(x1+1)%scount
                         x2=(x2+1)%scount
                         leng-=1
        s1 = dat[int(offset/8):int(((offset+size)/8) + 1)]
        s1 = self.listtostring(s1)
        s1 = s1[offset%8 : ] + s1[-((offset+size)%8):]
        return  s1



def readdir(self, path, fh):
    cl1 = getdata1(server,"child")
    parentpath=os.path.dirname(path)
          childpath=os.path.basename(path)
    for x in cl1:
        if x==path:
            return ['.', '..']+cl1[path]


def readlink(self, path):
          global readcheckserv, scount
    sum1=check_server_read()
          if sum1>=(scount-2):
              x=hash(path)%scount
              x1=(x+1)%scount
              x2=(x+2)%scount
        dat = getdata2(serv[x], path,x)
    return self.listtostring(dat)
```

```
def removexattr(self, path, name):
    fl1 = getdata1(server,"files")
    attrs = fl1[path].get('attrs', {})
    try:
        del attrs[name]
        putdata1(server,"files",fl1)
    except KeyError:
        pass



def rename(self, old, new):
          global serv, scount
    check_server()
    fl1 = getdata1(server,"files")
    cl1 = getdata1(server,"child")
          x = hash(old)%scount
          x1 = (x+1)%scount
          x2 = (x+2)%scount
          y = hash(new)%scount
          y1 = (y+1)%scount
          y2 = (y+2)%scount
          dat = getdata3(serv[x],x)
          parentpathold=os.path.dirname(old)
          childpathold=os.path.basename(old)
          parentpathnew=os.path.dirname(new)
          childpathnew=os.path.basename(new)
          if childpathold == childpathnew:
                        cl1[parentpathnew].append(childpathnew)
                        cl1[parentpathold].remove(childpathold)
                        for i in cl1.keys():
                                      listnew=i.split('/')
                                      if childpathold in listnew:
                                                    a_po=i.find(childpathold)
```

```
                attach=i[a_po:]
                newkey=parentpathnew + '/' + attach
                cl1[newkey]=cl1.pop(i)
                fl1[newkey]=fl1.pop(i)


for i in dat.keys():
        listnew=i.split('/')
        if childpathold in listnew:
                a_po=i.find(childpathold)
                attach=i[a_po:]
                newkey=parentpathnew + '/' + attach
                fl1[newkey]=fl1.pop(i)
                n=0
                x = hash(i)%scount
                x1 = (x+1)%scount
                x2 = (x+2)%scount
                y = hash(newkey)%scount
                y1 = (y+1)%scount
                y2 = (y+2)%scount
                while(n<scount):
                        temp=[]
                        temp1=[]
                        temp2=[]
                        temp3=[]
                        temp4=[]
                        temp5=[]
                dat = getdata3(serv[x], x)
                dat1 = r1getdata3(serv[x1], x1)
                dat2 = r2getdata3(serv[x2], x2)
                dat3 = cgetdata3(serv[x],x)
                dat4 = cr1getdata3(serv[x1],x1)
                dat5 = cr2getdata3(serv[x2],x2)
                        temp = dat[i]
        temp1 = dat1[i]
        temp2 = dat2[i]
        temp3 = dat3[i]
        temp4 = dat4[i]
        temp5 = dat5[i]
        dat.pop(i)
        dat1.pop(i)
        dat2.pop(i)
        dat3.pop(i)
        dat4.pop(i)
        dat5.pop(i)
        putdata3(serv[x], dat,x)
r1putdata3(serv[x1], dat1,x1)
r2putdata3(serv[x2], dat2,x2)
cputdata3(serv[x], dat3,x)
cr1putdata3(serv[x1], dat4,x1)
cr2putdata3(serv[x2], dat5,x2)

dat = getdata3(serv[y], y)
dat1 = r1getdata3(serv[y1], y1)
dat2 = r2getdata3(serv[y2], y2)
dat3 = cgetdata3(serv[y],y)
dat4 = cr1getdata3(serv[y1],y1)
dat5 = cr2getdata3(serv[y2],y2)
        dat[newkey]=temp
        dat1[newkey]=temp1
        dat2[newkey]=temp2
        dat3[newkey]=temp3
        dat4[newkey]=temp4
        dat5[newkey]=temp5
        putdata3(serv[y], dat,y)
r1putdata3(serv[y1], dat1,y1)
r2putdata3(serv[y2], dat2,y2)
cputdata3(serv[y], dat3,y)
```

```
            cr1putdata3(serv[y1], dat4,y1)
            cr2putdata3(serv[y2], dat5,y2)
                n+=1
                x=(x+1)%scount
                x1=(x1+1)%scount
                x2=(x2+1)%scount
                y=(y+1)%scount
                y1=(y1+1)%scount
                y2=(y2+1)%scount



else:
    for i in dat.keys():
        oldkey=i
        listnew=i.split('/')
        for n,k in enumerate(listnew):
            if k==childpathold:
                listnew[n]=childpathnew
                newkey = "/".join(listnew)
                fl1[newkey] =fl1.pop(oldkey)
                n=0
                x = hash(oldkey)%scount
                x1 = (x+1)%scount
                x2 = (x+2)%scount
                y = hash(newkey)%scount
                y1 = (y+1)%scount
                y2 = (y+2)%scount
                while(n<scount):
                    temp=[]
                    temp1=[]
                    temp2=[]
                    temp3=[]
                    temp4=[]
                    temp5=[]
```

```
dat = getdata3(serv[x], x)
dat1 = r1getdata3(serv[x1], x1)
dat2 = r2getdata3(serv[x2], x2)
dat3 = cgetdata3(serv[x],x)
dat4 = cr1getdata3(serv[x1],x1)
dat5 = cr2getdata3(serv[x2],x2)
        temp = dat[oldkey]
        temp1 = dat1[oldkey]
        temp2 = dat2[oldkey]
        temp3 = dat3[oldkey]
        temp4 = dat4[oldkey]
        temp5 = dat5[oldkey]
        dat.pop(oldkey)
        dat1.pop(oldkey)
        dat2.pop(oldkey)
        dat3.pop(oldkey)
        dat4.pop(oldkey)
        dat5.pop(oldkey)
        putdata3(serv[x], dat,x)
r1putdata3(serv[x1], dat1,x1)
r2putdata3(serv[x2], dat2,x2)
cputdata3(serv[x], dat3,x)
cr1putdata3(serv[x1], dat4,x1)
cr2putdata3(serv[x2], dat5,x2)

dat = getdata3(serv[y], y)
dat1 = r1getdata3(serv[y1], y1)
dat2 = r2getdata3(serv[y2], y2)
dat3 = cgetdata3(serv[y],y)
dat4 = cr1getdata3(serv[y1],y1)
dat5 = cr2getdata3(serv[y2],y2)
        dat[newkey]=temp
        dat1[newkey]=temp1
        dat2[newkey]=temp2
```

```
                        dat3[newkey]=temp3
                        dat4[newkey]=temp4
                        dat5[newkey]=temp5
                        putdata3(serv[y], dat,y)
                    r1putdata3(serv[y1], dat1,y1)
                    r2putdata3(serv[y2], dat2,y2)
                    cputdata3(serv[y], dat3,y)
                    cr1putdata3(serv[y1], dat4,y1)
                    cr2putdata3(serv[y2], dat5,y2)
                        n+=1
                        x=(x+1)%scount
                        x1=(x1+1)%scount
                        x2=(x2+1)%scount
                        y=(y+1)%scount
                        y1=(y1+1)%scount
                        y2=(y2+1)%scount

        for i in cl1.keys():
            oldkey=i
            listnew=i.split('/')
            for n,k in enumerate(listnew):
                if k==childpathold:
                    listnew[n]=childpathnew
                    newkey = "/".join(listnew)
                    cl1[newkey]=cl1.pop(oldkey)
                    fl1[newkey] =fl1.pop(oldkey)

        parentnon= os.path.dirname(new)
        cl1[parentnon].append(childpathnew)
        cl1[parentnon].remove(childpathold)

    putdata1(server,"files",fl1)
putdata1(server,"child",cl1)
```

```
def rmdir(self, path):
          global serv, scount
    check_server()
    fl1 = getdata1(server,"files")
    cl1 = getdata1(server,"child")
          if fl1[path]['st_nlink'] <= 2:
        parentpath=os.path.dirname(path)
              childpath=os.path.basename(path)
              fl1.pop(path)
        cl1.pop(path)
        cl1[parentpath].remove(childpath)
        fl1[parentpath]['st_nlink'] -= 1
        putdata1(server,"files",fl1)
        putdata1(server,"child",cl1)
          else:
              raise FuseOSError(ENOTEMPTY)

def setxattr(self, path, name, value, options, position=0):
    fl1 = getdata1(server,"files")
    attrs = fl1[path].setdefault('attrs', {})
    attrs[name] = value
    putdata1(server,"files",fl1)



def statfs(self, path):
    return dict(f_bsize=512, f_blocks=4096, f_bavail=2048)

def symlink(self, target, source):
          global serv, scount
    check_server()
    fl1 = getdata1(server,"files")
          cl1 = getdata1(server,"child")
    fl1[target] = dict(st_mode=(S_IFLNK | 0o777), st_nlink=1, st_size=len(source))
```

```
            parentpath=os.path.dirname(target)
            childpath=os.path.basename(target)
            cl1[parentpath].append(childpath)
        putdata1(server,"files",fl1)
            putdata1(server,"child",cl1)
            x=hash(target)%scount
            x1=(x+1)%scount
            x2=(x+2)%scount
            dat = getdata2(serv[x], target,x)
            dat = self.stringtolist(source)
        putdata2(serv[x], target, dat,x)
        r1putdata2(serv[x1], target, dat,x1)
        r2putdata2(serv[x2], target, dat,x2)
            cdat=hash(source)
        cputdata2(serv[x], target, dat,x)
        cr1putdata2(serv[x1], target, dat,x1)
        cr2putdata2(serv[x2], target, dat,x2)


    def truncate(self, path, length, fh=None):
            global serv, scount
        check_server()
            x=hash(path)%scount
            x1=(x+1)%scount
            x2=(x+2)%scount
        fl1 = getdata1(server,"files")
        leng= (fl1[path]['st_size'])
            if length > leng:
          fl1[path]['st_size'] = leng
          putdata1(server,"files",fl1)
            else:
              if leng%8 == 0:
                  leng=int(leng/8)
              else:
                    leng=int((leng/8)+1)
              element=0
              element_cond=0
              dat=[]
              r1dat=[]
              r2dat=[]
              while leng!=0:
                    data2=getdata2(serv[x], path,x)
                    r1data2=r1getdata2(serv[x1], path,x1)
                    r2data2=r2getdata2(serv[x2], path,x2)
                    dat.append(data2[element])
                    r1dat.append(r1data2[element])
                    r2dat.append(r2data2[element])
                    element_cond=(element_cond+1)%scount
                    if element_cond == 0:
                          element+=1
                    x=(x+1)%scount
                    x1=(x1+1)%scount
                    x2=(x2+1)%scount
                    leng-=1
          s1 = dat
          s1 = self.listtostring(s1)
              s1 = s1[0:length]
          r1s1 = r1dat
          r1s1 = self.listtostring(r1s1)
              r1s1 = r1s1[0:length]
          r2s1 = r2dat
          r2s1 = self.listtostring(r2s1)
              r2s1 = r2s1[0:length]
              i=0
          x=hash(path)%scount
          x1=(x+1)%scount
          x2=(x+2)%scount
              while i!=scount:
                dat = getdata3(serv[x],x)
                cdat = cgetdata3(serv[x],x)
                if path not in dat.keys():
                            i+=1
            x=(x+1)%scount
                            continue
dat.pop(path)
            putdata3(serv[x], dat,x)
cdat.pop(path)
            cputdata3(serv[x], cdat,x)
            i+=1
            x=(x+1)%scount
   i=0
   while i!=scount:
            r1dat = r1getdata3(serv[x1],x1)
            cr1dat = cr1getdata3(serv[x1],x1)
            if path not in r1dat.keys():
                        i+=1
        x1=(x1+1)%scount
                        continue
r1dat.pop(path)
            r1putdata3(serv[x1], r1dat,x1)
cr1dat.pop(path)
            cr1putdata3(serv[x1], cr1dat,x1)
            i+=1
            x1=(x1+1)%scount
   i=0
   while i!=scount:
            r2dat = r2getdata3(serv[x2],x2)
            cr2dat = cr2getdata3(serv[x2],x2)
            if path not in r2dat.keys():
                        i+=1
        x2=(x2+1)%scount
                        continue
```

```
  r2dat.pop(path)
              r2putdata3(serv[x2], r2dat,x2)
  cr2dat.pop(path)
              cr2putdata3(serv[x2], cr2dat,x2)
              i+=1
              x2=(x2+1)%scount
     leng=len(s1)
     if leng%8 == 0:
              leng=int(leng/8)
     else:
              leng=int((leng/8)+1)
     data1=self.stringtolist(s1)
x=hash(path)%scount
     i=0
     while leng!=0:
              dat = getdata2(serv[x],path,x)
  dat.append(data1[i])
              putdata2(serv[x],path,dat,x)
              cdat = cgetdata2(serv[x],path,x)
  cdat.append(hash(data1[i]))
              cputdata2(serv[x],path,cdat,x)
              x=(x+1)%scount
              i+=1
              leng-=1


     r1leng=len(r1s1)
     if r1leng%8 == 0:
              r1leng=int(r1leng/8)
     else:
              r1leng=int((r1leng/8)+1)
     r1data1=self.stringtolist(r1s1)
x=hash(path)%scount
     x1=(x+1)%scount
     i=0
```

```
        while r1leng!=0:
                r1dat = r1getdata2(serv[x1],path,x1)
    r1dat.append(r1data1[i])
                r1putdata2(serv[x1],path,r1dat,x1)
                cr1dat = cr1getdata2(serv[x1],path,x1)
    cr1dat.append(hash(r1data1[i]))
                cr1putdata2(serv[x1],path,cr1dat,x1)
                x1=(x1+1)%scount
                i+=1
                r1leng-=1



        r2leng=len(r2s1)
        if r2leng%8 == 0:
                r2leng=int(r2leng/8)
        else:
                r2leng=int((r2leng/8)+1)
        r2data1=self.stringtolist(r2s1)
x=hash(path)%scount
        x2=(x+2)%scount
        i=0
        while r2leng!=0:
                r2dat = r2getdata2(serv[x2],path,x2)
    r2dat.append(r2data1[i])
                r2putdata2(serv[x2],path,r2dat,x2)
                cr2dat = cr2getdata2(serv[x2],path,x2)
    cr2dat.append(hash(r2data1[i]))
                cr2putdata2(serv[x2],path,cr2dat,x2)
                x2=(x2+1)%scount
                i+=1
                r2leng-=1


fl1[path]['st_size'] = length
putdata1(server,"files",fl1)
```

```
def unlink(self, path):
          global serv, scount
    check_server()
    fl1 = getdata1(server,"files")
    cl1 = getdata1(server,"child")
          fl1.pop(path)
    parentpath=os.path.dirname(path)
          childpath=os.path.basename(path)
    cl1[parentpath].remove(childpath)
    putdata1(server,"files",fl1)
    putdata1(server,"child",cl1)
    x=hash(path)%scount
          x1=(x+1)%scount
          x2=(x+2)%scount
          i=0
          while i!=scount:
              dat = getdata3(serv[x],x)
              r1dat = r1getdata3(serv[x1],x1)
              r2dat = r2getdata3(serv[x2],x2)
        dat.pop(path)
        r1dat.pop(path)
        r2dat.pop(path)
              putdata3(serv[x], dat,x)
              r1putdata3(serv[x1], r1dat,x1)
              r2putdata3(serv[x2], r2dat,x2)
              cdat = cgetdata3(serv[x],x)
              cr1dat = cr1getdata3(serv[x1],x1)
              cr2dat = cr2getdata3(serv[x2],x2)
        cdat.pop(path)
        cr1dat.pop(path)
        cr2dat.pop(path)
              cputdata3(serv[x], cdat,x)
              cr1putdata3(serv[x1], cr1dat,x1)
              cr2putdata3(serv[x2], cr2dat,x2)
              i+=1
              x=(x+1)%scount
              x1=(x1+1)%scount
              x2=(x2+1)%scount



def utimens(self, path, times=None):
    fl1 = getdata1(server,"files")
    now = time()
    atime, mtime = times if times else (now, now)
    fl1[path]['st_atime'] = atime
    fl1[path]['st_mtime'] = mtime
    putdata1(server,"files",fl1)

def write(self, path, data, offset, fh):
          global serv, scount
    check_server()
    fl1 = getdata1(server,"files")
    leng=fl1[path]['st_size']
          if leng%8 == 0:
              leng=int(leng/8)
          else:
              leng=int((leng/8)+1)

    if leng==0:
        data1=self.stringtolist(data)
              x=hash(path)%scount
              x1=(x+1)%scount
              x2=(x+2)%scount
              length = len(data1)
              i=0
              while length!=0:
                    dat = getdata2(serv[x],path,x)
                    r1dat = r1getdata2(serv[x1],path,x1)
                    r2dat = r2getdata2(serv[x2],path,x2)
                    dat.append(data1[i])
                    r1dat.append(data1[i])
                    r2dat.append(data1[i])
                    putdata2(serv[x],path,dat,x)
                    r1putdata2(serv[x1],path,r1dat,x1)
                    r2putdata2(serv[x2],path,r2dat,x2)
                    cs = cgetdata2(serv[x],path,x)
                    r1cs = cr1getdata2(serv[x1],path,x1)
                    r2cs = cr2getdata2(serv[x2],path,x2)
                    cs.append(hash(data1[i]))
                    r1cs.append(hash(data1[i]))
                    r2cs.append(hash(data1[i]))
                    cputdata2(serv[x],path,cs,x)
                    cr1putdata2(serv[x1],path,r1cs,x1)
                    cr2putdata2(serv[x2],path,r2cs,x2)
                    x=(x+1)%scount
                    x1=(x1+1)%scount
                    x2=(x2+1)%scount
                    i+=1
                    length-=1

        else:
          x=hash(path)%scount
              x1=(x+1)%scount
              x2=(x+2)%scount
              data1=self.stringtolist(data)
              y=((leng+x)%scount)
              y1=((leng+x1)%scount)
              y2=((leng+x2)%scount)
              ll=getdata2(serv[y-1],path,(y-1))
              ll1=r1getdata2(serv[y1-1],path,(y1-1))
```

```
    ll2=r2getdata2(serv[y2-1],path,(y2-1))
    cll=cgetdata2(serv[y-1],path,(y-1))
    cll1=cr1getdata2(serv[y1-1],path,(y1-1))
    cll2=cr2getdata2(serv[y2-1],path,(y2-1))
if len(ll[-1]) == 8:
  i=0
      length = len(data1)
          while length!=0:
                  dat = getdata2(serv[y],path,y)
                  dat.append(data1[i])
                  putdata2(serv[y],path,dat,y)
                  cdat = cgetdata2(serv[y],path,y)
                  cdat.append(hash(data1[i]))
                  cputdata2(serv[y],path,cdat,y)
                  y=(y+1)%scount
                  i+=1
                  length-=1
if len(ll1[-1]) == 8:
  i=0
      length = len(data1)
          while length!=0:
                  r1dat = r1getdata2(serv[y1],path,y1)
                  r1dat.append(data1[i])
                  r1putdata2(serv[y1],path,r1dat,y1)
                  cr1dat = cr1getdata2(serv[y1],path,y1)
                  cr1dat.append(hash(data1[i]))
                  cr1putdata2(serv[y1],path,cr1dat,y1)
                  y1=(y1+1)%scount
                  i+=1
                  length-=1
if len(ll2[-1]) == 8:
  i=0
      length = len(data1)
          while length!=0:
                  r2dat = r2getdata2(serv[y2],path,y2)
                  r2dat.append(data1[i])
                  r2putdata2(serv[y2],path,r2dat,y2)
                  cr2dat = cr2getdata2(serv[y2],path,y2)
                  cr2dat.append(hash(data1[i]))
                  cr2putdata2(serv[y2],path,cr2dat,y2)
                  y2=(y2+1)%scount
                  i+=1
                  length-=1

if len(ll[-1])!=8:
  i=0
          y=y-1
          data1 = self.stringtolist(ll[-1] + data)
          del ll[-1]
          putdata2(serv[y],path,ll,y)
          del cll[-1]
          cputdata2(serv[y],path,cll,y)
          length=len(data1)
          while length!=0:
                  dat = getdata2(serv[y],path,y)
                  dat.append(data1[i])
                  putdata2(serv[y],path,dat,y)
                  cdat = cgetdata2(serv[y],path,y)
                  cdat.append(hash(data1[i]))
                  cputdata2(serv[y],path,cdat,y)
                  y=(y+1)%scount
                  i+=1
                  length-=1

if len(ll1[-1])!=8:
  i=0
          y1=y1-1
          data1 = self.stringtolist(ll1[-1] + data)
          del ll1[-1]
          r1putdata2(serv[y1],path,ll1,y1)
          del cll1[-1]
          cr1putdata2(serv[y1],path,cll1,y1)
          length=len(data1)
          while length!=0:
                  r1dat = r1getdata2(serv[y1],path,y1)
                  r1dat.append(data1[i])
                  r1putdata2(serv[y1],path,r1dat,y1)
                  cr1dat = cr1getdata2(serv[y1],path,y1)
                  cr1dat.append(hash(data1[i]))
                  cr1putdata2(serv[y1],path,cr1dat,y1)
                  y1=(y1+1)%scount
                  i+=1
                  length-=1

if len(ll2[-1])!=8:
  i=0
          y2=y2-1
          data1 = self.stringtolist(ll2[-1] + data)
          del ll2[-1]
          r2putdata2(serv[y2],path,ll2,y2)
          del cll2[-1]
          cr2putdata2(serv[y2],path,cll2,y2)
          length=len(data1)
          while length!=0:
                  r2dat = r2getdata2(serv[y2],path,y2)
                  r2dat.append(data1[i])
                  r2putdata2(serv[y2],path,r2dat,y2)
                  cr2dat = cr2getdata2(serv[y2],path,y2)
                  cr2dat.append(hash(data1[i]))
                  cr2putdata2(serv[y2],path,cr2dat,y2)
                  y2=(y2+1)%scount
                  i+=1
```

```
                                  length-=1

        fl1[path]['st_size'] += len(data)
        putdata1(server,"files",fl1)
        return len(data)

if __name__ == '__main__':
    mport=argv[2]
    global scount
    scount = (len(argv)-3)
    global ports
    ports=[]
    i=0
    while i<scount:
        ports.append(int(argv[i+3]))
        i+=1
    print (ports)
    server = xmlrpclib.ServerProxy("http://localhost:"+str(int(mport)))
    if len(argv) > 8:
        print('usage: %s <mountpoint>' % argv[0])
        exit(1)

    logging.basicConfig(level=logging.DEBUG)
    fuse = FUSE(Memory(), argv[1], foreground=True)
```