Design Implementation

The proposed code is for building a client server system. The file system stores data into the remote server instead of local computers. With the help of meta server and data server file, the data is sent to the server. The meta server is used to store the metadata of files and directories and the data server to store data of files.

To send or put data on server, the meta server port and data server port along with their addresses are used. The pickle function statement in the code is used to serialize and de-serialize object structure. And using xml rpc library is being used for the procedure call.

Approximately, all functions returning values must send that to server. Thus, we use metaserveradd in code to send it to server which has its path.

Pickle loads and pickle dumps are frequently used statements in code. Loads will provide with meta data and and pickle dumps returns the object as bytes.

The code has modified functions for putting and getting data from the servers. The test cases implemented are mentioned below:

1. Write in a file “one.txt”
   Executed Command: echo “12345678901234567890” > one.txt
   My Output: A text file in fusemount with the data given
2. Making directories
   Executed Command: mkdir one , mkdir two
   Creates two directories in root
3. Moving (Renaming) a directory from “one” to “two”
   Executed Command: mv one two
   Renames one to two
   (The directories under old address are also moved to new place.)
4. Removing directory
   Executed Command: rmdir one
   Removes one from the root
5. Unlink
   While removing or deleting the directory the unlinking is done.
   Executed Command : ll
   It removes the name and unlinks it from its address and reduces count.
6. Truncate
   Executed Command: truncate -s 5 one.txt
   Returns 5 characters from the first block of the text file.

CODE:

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

from __future__ import print_function, absolute_import, division
```

```
import logging
import xmlrpclib,pickle
from collections import defaultdict
from errno import ENOENT
from stat import S_IFDIR, S_IFLNK, S_IFREG
from sys import argv, exit
from time import time

from fuse import FUSE, FuseOSError, Operations, LoggingMixIn
size_block=8                                                    #size defined for
division of data into 8 bytes blocks.



if not hasattr(__builtins__, 'bytes'):
   bytes = str

class Memory(LoggingMixIn, Operations):
   'Example memory filesystem. Supports only one level of files.'

   def __init__(self,metaserverport,dataserverport):
      self.files = {}
         self.size_block=8                                      #The data now is to be
divided into eight bytes chunks
         self.data = defaultdict(list)                          # data will be in the
format of elements of list
      self.fd = 0                                               #So, we give self.data to be a
(list) form
         self.metaserverport=metaserverport
         self.dataserverport=dataserverport
         print(self.metaserverport)
```

```
        print(self.dataserverport)
        self.metaserveradd=xmlrpclib.ServerProxy('http://localhost:' + str(self.metaserverport) + '/')

        self.dataserveradd=[]
        for i in range (0,len(self.dataserverport)):
                self.dataserveradd.append(xmlrpclib.ServerProxy('http://localhost:' + str(self.dataserverport[i]) + '/'))
        print (self.metaserveradd)
        print (self.dataserveradd)
    now = time()
        #self.files['/'] = dict(st_mode=(S_IFDIR | 0o755), st_ctime=now,st_mtime=now, st_atime=now, st_nlink=2,files=[]#here a new list is added to store the directory contents or say file contents.
        self.metaserveradd.put('/',pickle.dumps(dict(st_mode=(S_IFDIR | 0o755), st_ctime=now,st_mtime=now, st_atime=now, st_nlink=2, files = [])))
        print ('###PICKLE###')
        print(pickle.loads(self.metaserveradd.get('/')))

  def chmod(self, path, mode):
        metadata=pickle.loads(self.metaserveradd.get(path))
        metadata['st_mode'] &= 0o770000
    metadata['st_mode'] |= mode
        self.metaserveradd.put(path,pickle.dumps(metadata))
    return 0

  def chown(self, path, uid, gid):
        metadata=pickle.loads(self.metaserveradd.get(path))
    metadata['st_uid'] = uid
    metadata['st_gid'] = gid
        self.metaserveradd.put(path,pickle.dumps(metadata))
```

```
    def create(self, path, mode):
                print ("in mode...")
                print (mode)
        metadata = dict(st_mode=(S_IFREG | mode), st_nlink=1,st_size=0, st_ctime=time(), st_mtime=time(),st_atime=time(),files=[],blocks=[])
                self.metaserveradd.put(path,pickle.dumps(metadata))
                parent, child=self.dividepath(path)                    #dividepath function is called and path is divided into child and parent path
                                                                       #making a new file or directory appends it in the parent path which writes metadata to the parent path
                metadata = pickle.loads(self.metaserveradd.get(parent))
                metadata['files'].append(child)
                self.metaserveradd.put(parent,pickle.dumps(metadata))
                #print (self.files[path]['st_size'])
                self.fd += 1
        return self.fd


    def getattr(self, path, fh=None):
      if self.metaserveradd.get(path) == -1:
        raise FuseOSError(ENOENT)
          return pickle.loads(self.metaserveradd.get(path))


    def getxattr(self, path, name, position=0):
          if self.metaserveradd.get(path) == -1:
        return ''                                                      # Should return ENOATTR
          metadata = pickle.loads(self.metaserveradd.get(path))
          attrs = metadata.get('attrs', {})
          try:
        return attrs[name]
```

```
        except KeyError:
            return ''       # Should return ENOATTR

    def listxattr(self, path):
            if self.metaserveradd.get(path) == -1:
            return ''   # Should return ENOATTR
        metadata = pickle.loads(self.metaserveradd.get(path))
        attrs = metadata.get('attrs', {})
        return attrs.keys()

    def mkdir(self, path, mode):
        metadata = dict(st_mode=(S_IFDIR | mode), st_nlink=2,st_size=0, st_ctime=time(),
st_mtime=time(),st_atime=time(),files=[])
            self.metaserveradd.put(path,pickle.dumps(metadata))
            parent,child=self.dividepath(path)
            metadata=pickle.loads(self.metaserveradd.get(parent))                    #mkdir
appends the new directory to the parent path and adds metadata to the parent path
            metadata['st_nlink'] += 1
            metadata['files'].append(child)
            #print (first['files'])
            #metadata=pickle.loads(self.metaserveradd.get(child))
            self.metaserveradd.put(parent,pickle.dumps(metadata))
            #self.files['/']['st_nlink'] += 1

    def dividepath(self, path):
            child = path[path.rfind('/')+1:]                                    #dividepath     function
divides the path into the parent and child path
            parent = path[:path.rfind('/')]
            if parent == '':
                    parent='/'
```

```
            return parent,child


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


                                                                                    #           "Read"
subroutine reading the file after the data given was manipulated.
    def read(self, path, size, offset, fh):
            metadata = pickle.loads(self.metaserveradd.get(path))
        #new_string = ''.join([offset//size_block : (offset + size - 1)//size_block])
        #new_string = new_string[offset % size_block:offset % size_block + size]
            #metadata=pickle.loads(self.metaserveradd.get(path))
            #data=self.readdata(self,p['blocks'])
            data = self.readdata(path,metadata['blocks'])
        return data[offset:offset+size]


    def readdata(self,path,blocks):
        result = ''
        for i in range(0,len(blocks)):
            result += self.dataserveradd[blocks[i]].get(path + str(i))
        return result


    def readdir(self, path, fh):
            metadata=pickle.loads(self.metaserveradd.get(path))


            print (metadata)
        return ['.', '..'] + [x for x in metadata['files']]
```

```
def readlink(self, path):
        p=pickle.loads(self.metaserveradd.get(path))
        data=self.readdata(path,p['blocks'])
        return data[offset:offset+size]
    return data

def removexattr(self, path, name):
        if self.metaserveradd.get(path) == -1:
       return ''   # Should return ENOATTR
    metadata = pickle.loads(self.metaserveradd.get(path))
    attrs = metadata.get('attrs', {})

    try:
       del attrs[name]
       metadata.set('attrs', attrs)
       self.metaserveradd.put(path,pickle.dumps(metadata))
    except KeyError:
       pass       # Should return ENOATTR

def rename(self, old, new):
        op,oc=self.dividepath(old)
        np,nc=self.dividepath(new)
        #of=self.files[old]
        #cm=self.files[op]
        #npp = self.files[np]
        metadata=pickle.loads(self.metaserveradd.get(old))
        if metadata['st_mode'] & 0770000 == S_IFDIR:
        #checking here if it is a directory or file.
```

```
                self.mkdir(new,S_IFDIR)                                    # If it is
a directory, create a new directory using mkdir

                for f in metadata['files']:

                        #print (metadata['files'])

                        #print (f)

                        #self.files[new]['st_nlink']=self.files[old]['st_nlink']          #update  data
and meta data of old directory to the parent path of new path

                        #self.files[new]['files']=self.files[old]['files']          # with the help of
looping moving the entire directory(internal files/dir of it)into the new parent path location

                        print ('----------inside loop--------')

                        self.rename(old + '/' + f, new + '/' + f)

                self.rmdir(old)                                                    #removing the
old directory and path from old parent

        else:

                self.create(new,S_IFREG)
            #creating a new file at the new parent location

                self.files[np]['st_size']=self.files[op]['st_size']                # adding the meta data
and data to the new file

                data = self.data[oc]

                self.data[nc]=data

                self.files[op]['files'].remove(oc)                                 #removing the old file
from the old locatio

        #self.files[new] = self.files.pop(old)



    def rmdir(self, path):

            metadata=self.metaserveradd.pop_out(path)

            parent,child=self.dividepath(path)

            metadata=pickle.loads(self.metaserveradd.get(parent))

            #self.metaserveradd.put(parent,pickle.dumps(metadata))

            #first=self.files[parent]
```

```
        #print ('___')
        #print (first['files'])
        #parent_path=self.files[parent]
        metadata['files'].remove(child)
        metadata['st_nlink'] -=1
        self.metaserveradd.put(parent,pickle.dumps(metadata))
    #self.files['/']['st_nlink'] -= 1

def setxattr(self, path, name, value, options, position=0):
    # Ignore options
        if self.metaserveradd.get(path) == -1:
       return ''   # Should return ENOATTR
    metadata = pickle.loads(self.metaserveradd.get(path))
    attrs = metadata.setdefault('attrs', {})
    attrs[name] = value
    metadata.set('attrs', attrs)
    self.metaserveradd.put(path,pickle.dumps(metadata))


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

def symlink(self, target, source):
    self.files[target] = dict(st_mode=(S_IFLNK | 0o777), st_nlink=1,
                              st_size=len(source))
    d1 =target[target.rfind('/')+1:]
    self.data[target] = [source[i:i+size_block] for i in range(0, len(source), size_block)]
    self.data[target] = source
```

```
def truncate(self, path, length, fh=None):
        data=[]
        metadata = pickle.loads(self.metaserveradd.get(path))
        blocks=metadata['blocks']
        blocklen=length//size_block
        stringlen=length%size_block
        for i in range(0,len(blocks)):
                firstdata=self.dataserveradd[blocks[i]].get(path + str(i))
                data = data + [''.join(firstdata)]
        if length<metadata['st_size']:
                secdata=data[:blocklen]+[''.join(data[blocklen][:stringlen])]

        #self.newfiledata(path,path,metadata)
        #if length>metadata['st_size']:
        #       length1=length-metadata['st_size']
        #       for i in range (0,length1):
        #               listz=[]
        #               listz.append('0')
        #       listz=''.join(listz)
        #       secdata=data.extend(listz)

        offset=metadata['st_size']
    metadata['st_size'] = length
        metadata['blocks'] = blocks
        for i in range (0,len(secdata)):
                self.dataserveradd[blocks[i]].put(path+str(i),secdata[i])
        self.metaserveradd.put(path,pickle.dumps(metadata))
```

```
    def unlink(self, path):
            parent,child=self.dividepath(path)
            metadata=pickle.loads(self.metaserveradd.get(parent))
            #self.files has meta data along with files in it
            metadata['files'].remove(child)
            self.metaserveradd.put(parent,pickle.dumps(metadata))                    #removing the
child path and hence unlinking from hierarchy
        #self.files.pop(path)


    def utimens(self, path, times=None):
        now = time()
        atime, mtime = times if times else (now, now)
            metadata=pickle.loads(self.metaserveradd.get(path))
        metadata['st_atime'] = atime
        metadata['st_mtime'] = mtime



    def write(self, path, data, offset, fh):
            newDataInBlocks = []
        blocks = []
            x = hash(path)
        metadata = pickle.loads(self.metaserveradd.get(path))
        if len(metadata['blocks']) == 0:
          oldData = ''
        else:
          oldData = self.readdata(path,metaData['blocks'])
        newData = oldData[:offset].ljust(offset,'\x00') + data + oldData[offset + len(data):]
        n = 1
        for a in range(0,len(newData), self.size_block):
```

```
            newDataInBlocks.append(newData[a : a + self.size_block])
            blocks.append((x + n - 1) % len(self.dataserverport))
            n += 1;

              for i in range(0,len(newDataInBlocks)):
                print("infor")
                print("sdadasdasdsadsa" + str(blocks[i]) + " newdata" + str(newDataInBlocks[i]) + "iiiiiii" +
str(i))
                print("index")
                print (i)
                print (self.dataserveradd)
            self.dataserveradd[blocks[i]].put(path + str(i),newDataInBlocks[i])

        #self.writeData(path,newDataInBlocks,blocks)

        metadata['st_size'] = len(newData)
        metadata['blocks'] = blocks
        self.metaserveradd.put(path,pickle.dumps(metadata))
        return len(data)

    def writeData(self,path,newDataInBlocks,blocks):
              print (newDataInBlocks)
              print (blocks[0])
        for i in range(0,len(newDataInBlocks)-1):
            self.dataserveradd[blocks[i]].put(path + str(i),newDataInBlocks[i])

if __name__ == '__main__':
    if len(argv) < 4:
        print('usage: %s <mountpoint> <metaserver port> <dataserver port>' % argv[0])
```

```
    exit(1)
metaserverport=int(argv[2])
dataserverport=[]
for i in range(3,len(argv)):
        dataserverport.append(int(argv[i]))

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