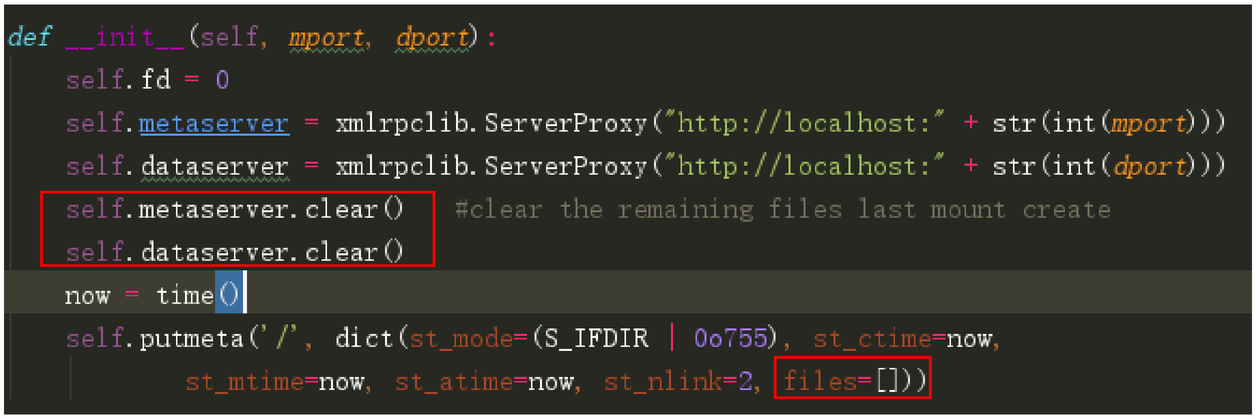
Homework 4 Design

Implementation:

In this homework, the dataserver.py and metaserver.py is the same as simpleht.py as given.

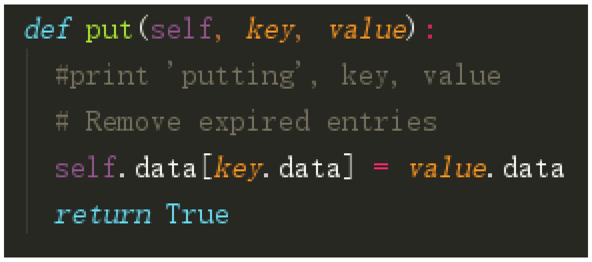
The main implementation is in remoteFS.py.

First of all, we need to connect to the meta server and the data server using xmlrpclib by the port number given and initialize so that both servers are empty and root metadata is stored.



```python
def __init__(self, mport, dport):
    self.fd = 0
    self.metaserver = xmlrpclib.ServerProxy("http://localhost:" + str(int(mport)))
    self.dataserver = xmlrpclib.ServerProxy("http://localhost:" + str(int(dport)))
    self.metaserver.clear()  #clear the remaining files last mount create
    self.dataserver.clear()
    now = time()
    self.putmeta('/', dict(st_mode=(S_IFDIR | 0o755), st_ctime=now,
            st_mtime=now, st_atime=now, st_nlink=2, files=[]))
```

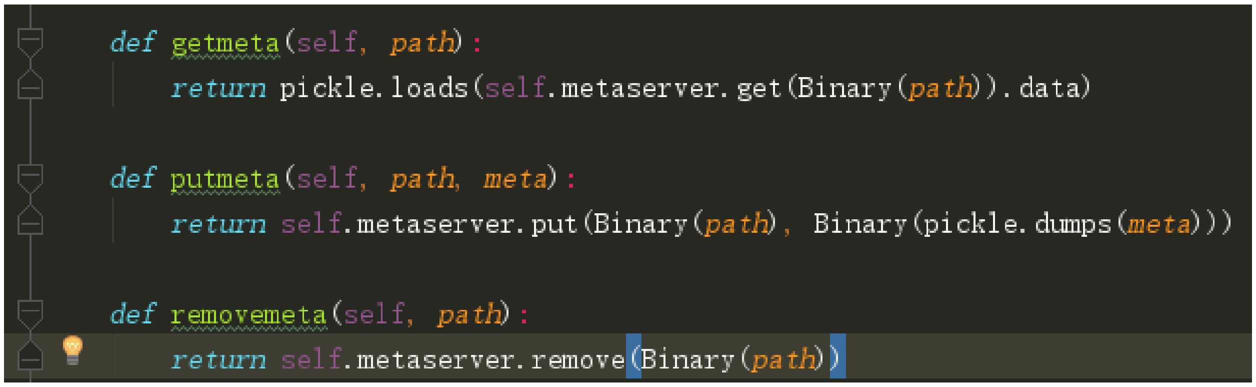
In this case, notice the method put in server is



```python
def put(self, key, value):
    #print 'putting', key, value
    # Remove expired entries
    self.data[key.data] = value.data
    return True
```

We need to store every single path in the map self.data in server. Therefore, we only need a list instead of dict to store the name of directories and files in the specific directory and create a new path pointing to the directories and files.

Put, get and remove metadata in meta server is easy, just call the put get and remove method in metaserver.py like this.

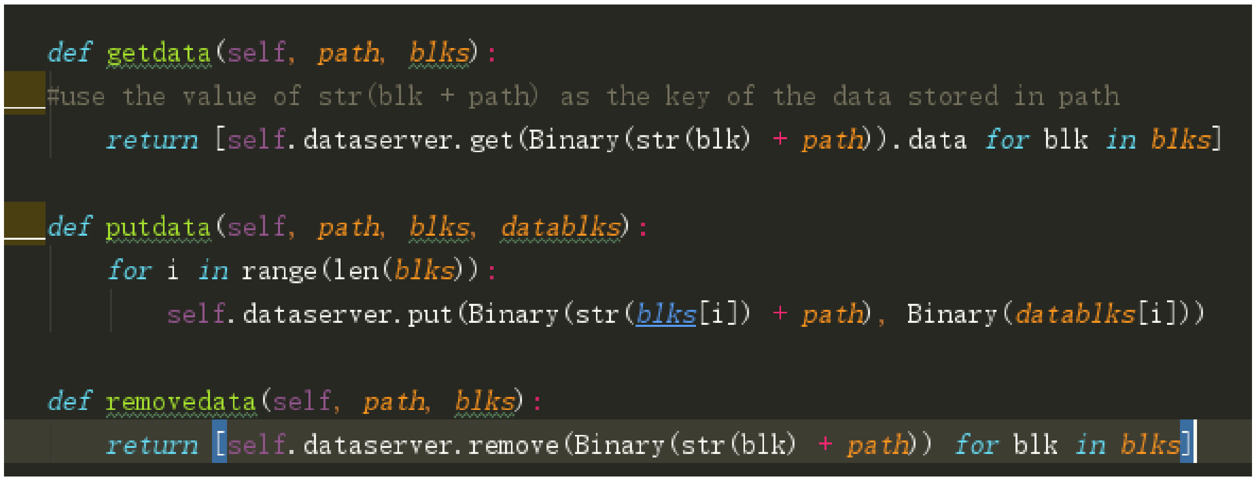


```
def getmeta(self, path):
    return pickle.loads(self.metaserver.get(Binary(path)).data)

def putmeta(self, path, meta):
    return self.metaserver.put(Binary(path), Binary(pickle.dumps(meta)))

def removemeta(self, path):
    return self.metaserver.remove(Binary(path))
```

Get, put and remove data in data server is a little tricky.



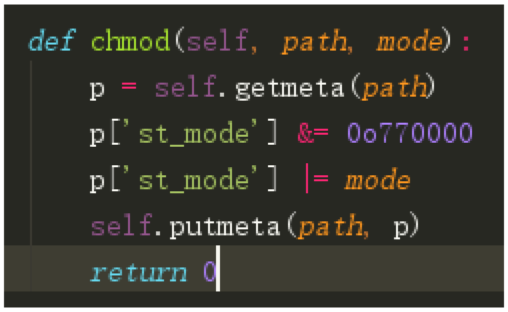
```
def getdata(self, path, blks):
#use the value of str(blk + path) as the key of the data stored in path
    return [self.dataserver.get(Binary(str(blk) + path)).data for blk in blks]

def putdata(self, path, blks, datablks):
    for i in range(len(blks)):
        self.dataserver.put(Binary(str(blks[i]) + path), Binary(datablks[i]))

def removedata(self, path, blks):
    return [self.dataserver.remove(Binary(str(blk) + path)) for blk in blks]
```

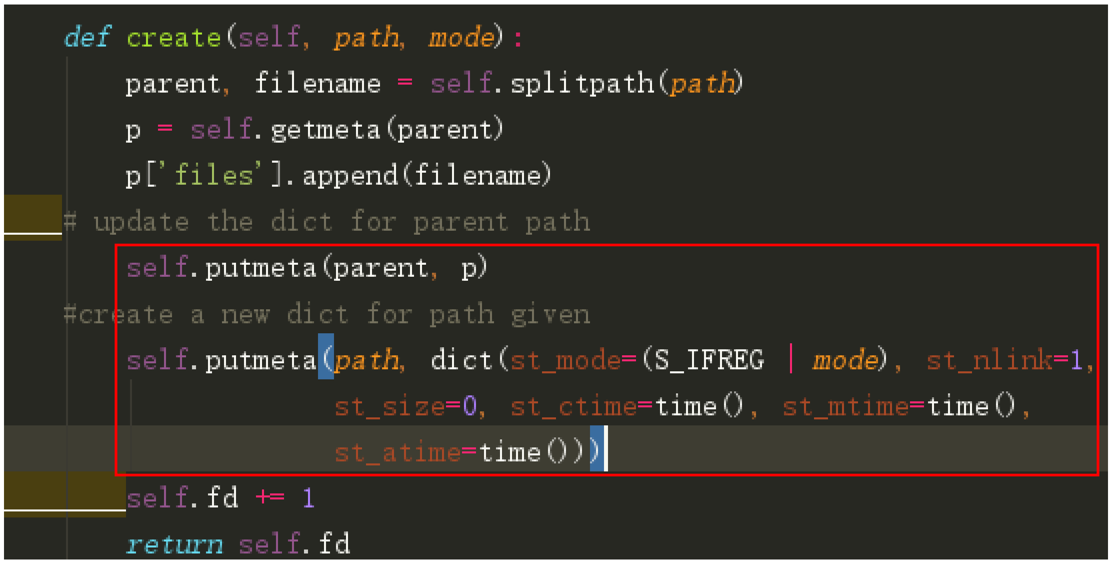
In this case, we use the string value of (blk number + path) as the key of one block data stored in data server. Therefore, when we deal with data in data server, just to iterate the blk numbers we need and call get, put, remove method in dataserver.py to do this operations block by block.

To modify the meta data for a file or directory is just similar as the last homework, we get meta data from the server, modify it and put new meta data to the server. The example is here.



```
def chmod(self, path, mode):
    p = self.getmeta(path)
    p['st_mode'] &= 0o770000
    p['st_mode'] |= mode
    self.putmeta(path, p)
    return 0
```

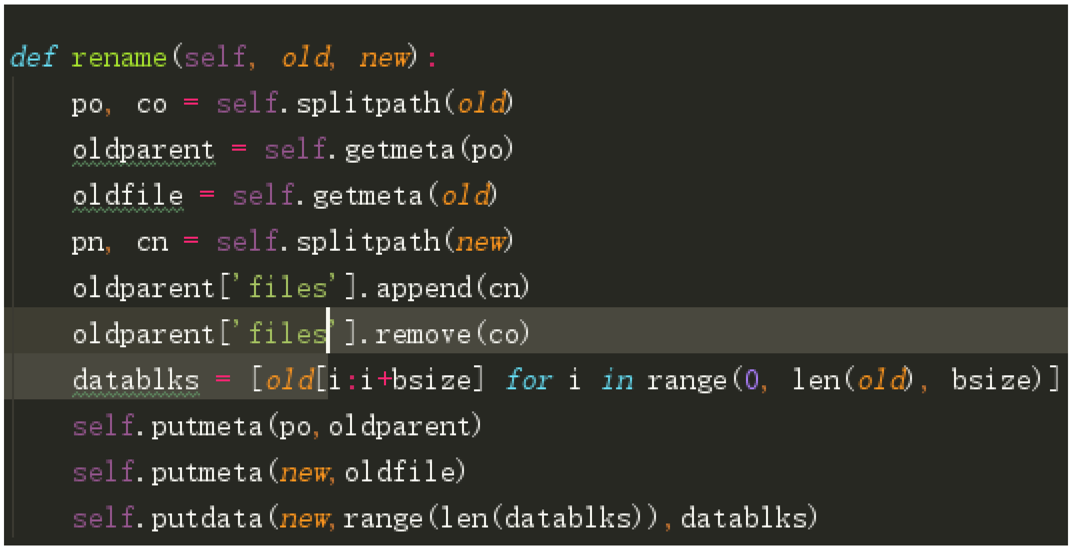
The create and mkdir is a little tricky. As I mentioned before, this hierarchy file system is actually “flat” in server part. When we create a file or directory. We need 2 put to update in the server.



```
    def create(self, path, mode):
        parent, filename = self.splitpath(path)
        p = self.getmeta(parent)
        p['files'].append(filename)
    # update the dict for parent path
        self.putmeta(parent, p)
    #create a new dict for path given
        self.putmeta(path, dict(st_mode=(S_IFREG | mode), st_nlink=1,
                          st_size=0, st_ctime=time(), st_mtime=time(),
                          st_atime=time()))
        self.fd += 1
        return self.fd
```

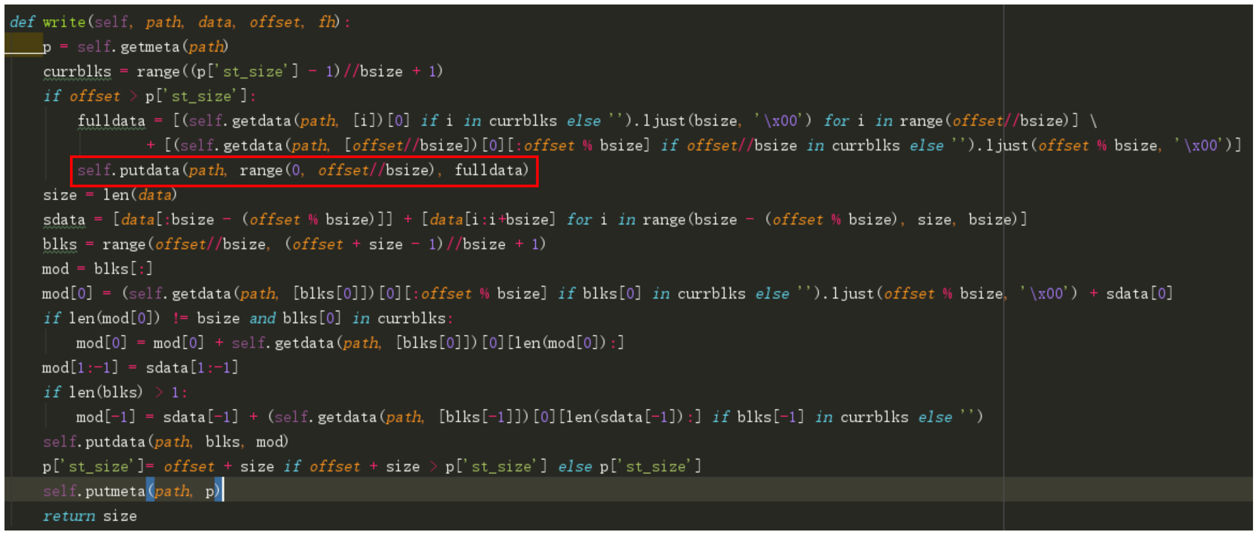
First is to add a file or directory name in the parent path ‘files’ list, then create a new dict in self.data for the created file or directory.

When we rename a file or a directory, we need to add new file name to parent and remove old name from parent, this operation will not change ‘st_nlink’ this time. Then we need to copy the old file data to a new address.



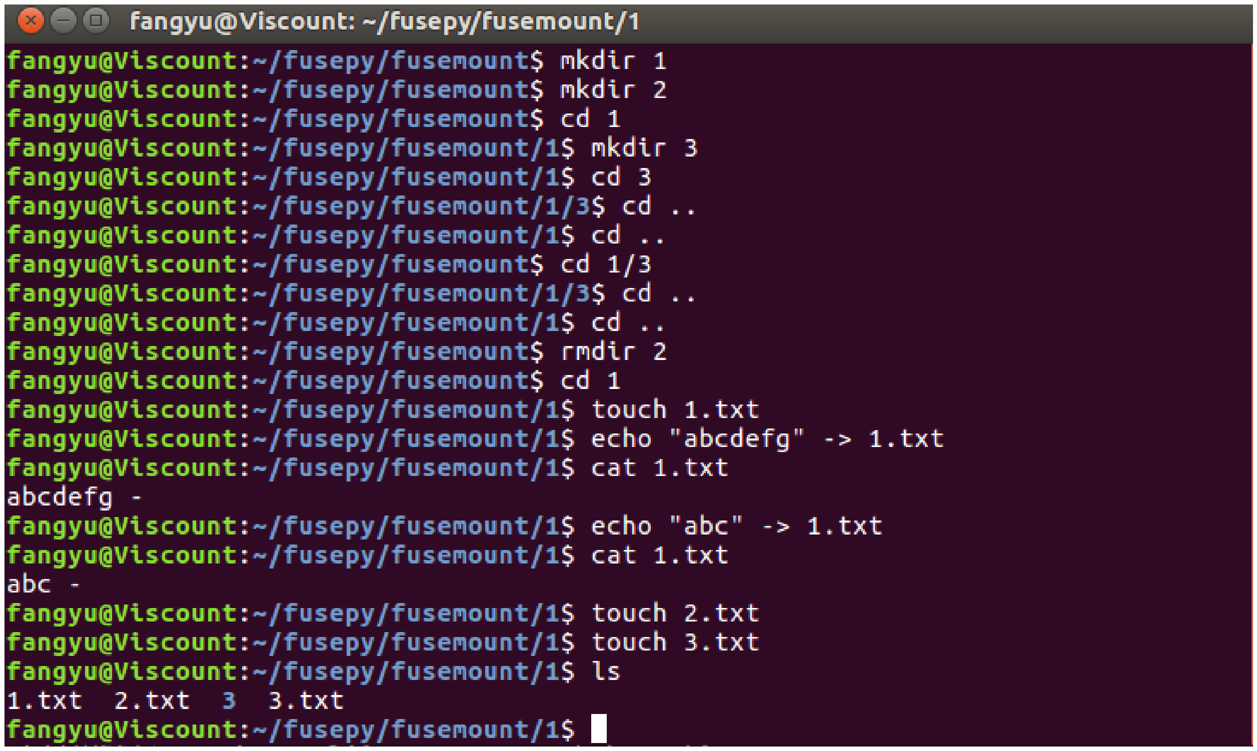
```
def rename(self, old, new):
    po, co = self.splitpath(old)
    oldparent = self.getmeta(po)
    oldfile = self.getmeta(old)
    pn, cn = self.splitpath(new)
    oldparent['files'].append(cn)
    oldparent['files'].remove(co)
    datablks = [old[i:i+bsize] for i in range(0, len(old), bsize)]
    self.putmeta(po,oldparent)
    self.putmeta(new,oldfile)
    self.putdata(new,range(len(datablks)),datablks)
```

When we read or write data in a file, it’s very similar to the last time. The only different is to give the block numbers we want to modify as a parameter in putdata method

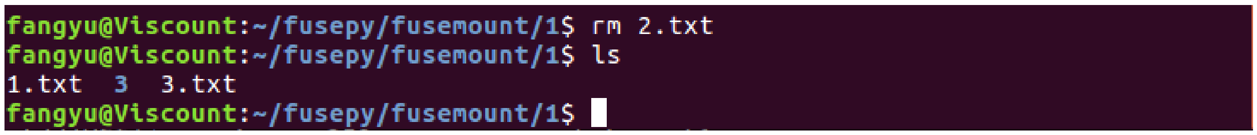


```python
def write(self, path, data, offset, fh):
    p = self.getmeta(path)
    currblks = range((p['st_size'] - 1)//bsize + 1)
    if offset > p['st_size']:
        fulldata = [(self.getdata(path, [i])[0] if i in currblks else '').ljust(bsize, '\x00') for i in range(offset//bsize)] \
                + [(self.getdata(path, [offset//bsize])[0][:offset % bsize] if offset//bsize in currblks else '').ljust(offset % bsize, '\x00')]
        self.putdata(path, range(0, offset//bsize), fulldata)
    size = len(data)
    sdata = [data[:bsize - (offset % bsize)]] + [data[i:i+bsize] for i in range(bsize - (offset % bsize), size, bsize)]
    blks = range(offset//bsize, (offset + size - 1)//bsize + 1)
    mod = blks[:]
    mod[0] = (self.getdata(path, [blks[0]])[0][:offset % bsize] if blks[0] in currblks else '').ljust(offset % bsize, '\x00') + sdata[0]
    if len(mod[0]) != bsize and blks[0] in currblks:
        mod[0] = mod[0] + self.getdata(path, [blks[0]])[0][len(mod[0]):]
    mod[1:-1] = sdata[1:-1]
    if len(blks) > 1:
        mod[-1] = sdata[-1] + (self.getdata(path, [blks[-1]])[0][len(sdata[-1]):] if blks[-1] in currblks else '')
    self.putdata(path, blks, mod)
    p['st_size']= offset + size if offset + size > p['st_size'] else p['st_size']
    self.putmeta(path, p)
    return size
```

Test

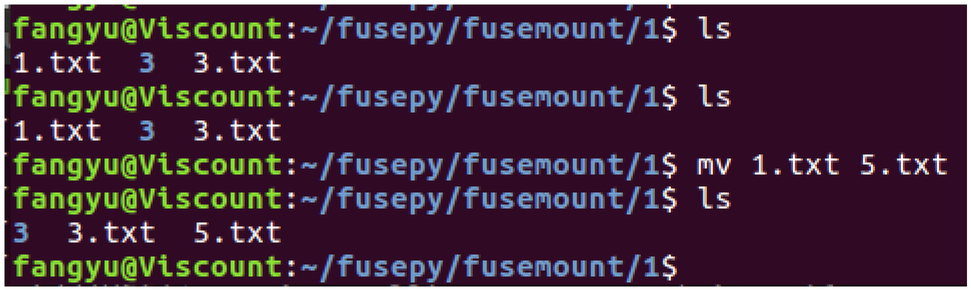


```
fangyu@Viscount: ~/fusepy/fusemount/1
fangyu@Viscount:~/fusepy/fusemount$ mkdir 1
fangyu@Viscount:~/fusepy/fusemount$ mkdir 2
fangyu@Viscount:~/fusepy/fusemount$ cd 1
fangyu@Viscount:~/fusepy/fusemount/1$ mkdir 3
fangyu@Viscount:~/fusepy/fusemount/1$ cd 3
fangyu@Viscount:~/fusepy/fusemount/1/3$ cd ..
fangyu@Viscount:~/fusepy/fusemount/1$ cd ..
fangyu@Viscount:~/fusepy/fusemount$ cd 1/3
fangyu@Viscount:~/fusepy/fusemount/1/3$ cd ..
fangyu@Viscount:~/fusepy/fusemount/1$ cd ..
fangyu@Viscount:~/fusepy/fusemount$ rmdir 2
fangyu@Viscount:~/fusepy/fusemount$ cd 1
fangyu@Viscount:~/fusepy/fusemount/1$ touch 1.txt
fangyu@Viscount:~/fusepy/fusemount/1$ echo "abcdefg" -> 1.txt
fangyu@Viscount:~/fusepy/fusemount/1$ cat 1.txt
abcdefg -
fangyu@Viscount:~/fusepy/fusemount/1$ echo "abc" -> 1.txt
fangyu@Viscount:~/fusepy/fusemount/1$ cat 1.txt
abc -
fangyu@Viscount:~/fusepy/fusemount/1$ touch 2.txt
fangyu@Viscount:~/fusepy/fusemount/1$ touch 3.txt
fangyu@Viscount:~/fusepy/fusemount/1$ ls
1.txt  2.txt  3  3.txt
fangyu@Viscount:~/fusepy/fusemount/1$ 
```



```
fangyu@Viscount:~/fusepy/fusemount/1$ rm 2.txt
fangyu@Viscount:~/fusepy/fusemount/1$ ls
1.txt  3  3.txt
fangyu@Viscount:~/fusepy/fusemount/1$ 
```

This test was to test the basic operation of the RPC file system like make, remove directories files, write, read, truncate files.



```
fangyu@Viscount:~/fusepy/fusemount/1$ ls
1.txt  3  3.txt
fangyu@Viscount:~/fusepy/fusemount/1$ ls
1.txt  3  3.txt
fangyu@Viscount:~/fusepy/fusemount/1$ mv 1.txt 5.txt
fangyu@Viscount:~/fusepy/fusemount/1$ ls
3  3.txt  5.txt
fangyu@Viscount:~/fusepy/fusemount/1$
```

This was to test the rename method.

Code

```python
import logging, xmlrpclib, pickle

from xmlrpclib import Binary
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 fuse import FUSE, FuseOSError, Operations, LoggingMixIn

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

bsize = 8

class Memory(LoggingMixIn, Operations):
    """RPC file system"""

    def __init__(self, mport, dport):
        self.fd = 0
        self.metaserver = xmlrpclib.ServerProxy("http://localhost:" +
str(int(mport)))
        self.dataserver = xmlrpclib.ServerProxy("http://localhost:" +
str(int(dport)))
        self.metaserver.clear()   #clear the remaining files last mount create
        self.dataserver.clear()
        now = time()
        self.putmeta('/', dict(st_mode=(S_IFDIR | 0o755), st_ctime=now,
                st_mtime=now, st_atime=now, st_nlink=2, files=[]))

    def splitpath(self, path):
        filename = path[path.rfind('/')+1:]
        parentpath = path[:path.rfind('/')]
        if parentpath == '':
            parentpath = '/'
        return parentpath, filename


    def getmeta(self, path):
        return pickle.loads(self.metaserver.get(Binary(path)).data)

    def putmeta(self, path, meta):
        return self.metaserver.put(Binary(path), Binary(pickle.dumps(meta)))

    def removemeta(self, path):
        return self.metaserver.remove(Binary(path))

    def getdata(self, path, blks):
   #use the value of str(blk + path) as the key of the data stored in path
        return [self.dataserver.get(Binary(str(blk) + path)).data for blk in
blks]

    def putdata(self, path, blks, datablks):
        for i in range(len(blks)):
            self.dataserver.put(Binary(str(blks[i]) + path), Binary(datablks[i]))

    def removedata(self, path, blks):
        return [self.dataserver.remove(Binary(str(blk) + path)) for blk in blks]


    def chmod(self, path, mode):
        p = self.getmeta(path)
        p['st_mode'] &= 0o770000
        p['st_mode'] |= mode
        self.putmeta(path, p)
        return 0

    def chown(self, path, uid, gid):
        p = self.getmeta(path)
        p['st_uid'] = uid
        p['st_gid'] = gid
        self.putmeta(path)

    def create(self, path, mode):
        parent, filename = self.splitpath(path)
        p = self.getmeta(parent)
        p['files'].append(filename)
  # update the dict for parent path
        self.putmeta(parent, p)
  #create a new dict for path given
        self.putmeta(path, dict(st_mode=(S_IFREG | mode), st_nlink=1,
                     st_size=0, st_ctime=time(), st_mtime=time(),
                     st_atime=time()))
        self.fd += 1
        return self.fd

    def getattr(self, path, fh = None):
        try:
            p = self.getmeta(path)
        except:
            raise FuseOSError(ENOENT)
        return {attr:p[attr] for attr in p.keys() if attr != 'files'}

    def getxattr(self, path, name, position=0):
        p = self.getmeta(path)
        attrs = p.get('attrs', {})
        try:
            return attrs[name]
        except KeyError:
            return ''        # Should return ENOATTR

    def listxattr(self, path):
        p = self.getmeta(path)
        attrs = p.get('attrs', {})
        return attrs.keys()
    def mkdir(self, path, mode):
        parent, dirname = self.splitpath(path)
        p = self.getmeta(parent)
        p['files'].append(dirname)
        p['st_nlink'] += 1
        self.putmeta(parent, p)
        self.putmeta(path, dict(st_mode=(S_IFDIR | mode), st_nlink=2,
                                st_size=0, st_ctime=time(), st_mtime=time(),
                                st_atime=time(),files=[]))

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

    def read(self, path, size, offset, fh):
        p = self.getmeta(path)
        if offset + size > p['st_size']:
            size = p['st_size'] - offset
        dd = ''.join(self.getdata(path, range(offset//bsize, (offset + size -
1)//bsize + 1)))
        dd = dd[offset % bsize:offset % bsize + size]
        return dd

    def readdir(self, path, fh):
        p = self.getmeta(path)
        return ['.', '..'] + p['files']

    def readlink(self, path):
        p = self.getmeta(path)
        return ''.join(self.getdata(path, range(p['st_size']//bsize)))

    def removexattr(self, path, name):
        p = self.getmeta(path)
        attrs = p.get('attrs', {})
        try:
            del attrs[name]
        except KeyError:
            pass        # Should return ENOATTR
        self.putmeta(path, p)

    def rename(self, old, new):
        po, co = self.splitpath(old)
        oldparent = self.getmeta(po)
        oldfile = self.getmeta(old)
        pn, cn = self.splitpath(new)
        oldparent['files'].append(cn)
        oldparent['files'].remove(co)
        datablks = [old[i:i+bsize] for i in range(0, len(old), bsize)]
        self.putmeta(po,oldparent)
        self.putmeta(new,oldfile)
        self.putdata(new,range(len(datablks)),datablks)


#   self.removemeta(old)


    def rmdir(self, path):
        p = self.getmeta(path)
        if len(p['files']) > 0:
            raise FuseOSError(ENOTEMPTY)
        self.removemeta(path)
        parent, dirname = self.splitpath(path)
        p = self.getmeta(parent)
        p['files'].remove(dirname)
        p['st_nlink'] -= 1
        self.putmeta(parent, p)

    def setxattr(self, path, name, value, options, position=0):
        # Ignore options
        p = self.getmeta(path)
        attrs = p.setdefault('attrs', {})
        attrs[name] = value

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

    def symlink(self, target, source):
        parent, filename = self.splitpath(target)
        p = self.getmeta(parent)
        p['files'].append(filename)
        self.putmeta(parent, p)
        self.putmeta(target, dict(st_mode=(S_IFLNK | 0o777), st_nlink=1,
                                  st_size=len(source)))
        datablks = [source[i:i+bsize] for i in range(0, len(source), bsize)]
        self.putdata(target, range(len(datablks)), datablks)
    def truncate(self, path, length, fh=None):
       p = self.getmeta(path)
        currblks = range((p[’st_size’] - 1)//bsize + 1)
        newblks  = range((length - 1)//bsize + 1)
        # create new blocks with the truncate length and remove the old blocks
        createblks = list(set(newblks[:-1]) - set(currblks))
        self.putdata(path, createblks, [’\x00’*bsize]*len(createblks))
        removeblks = list(set(currblks) - set(newblks))
        self.removedata(path, removeblks)
        # do truncate
        if len(newblks) > 0:
            if newblks[-1] in currblks:
                self.putdata(path, [newblks[-1]], [self.getdata(path, [newblks[-
1]])[0][:length % offset]])
            else:
                self.putdata(path, [newblks[-1]], [’\x00’*(length % bsize)])
        p = self.getmeta(path)
        p[’st_size’] = length
        self.putmeta(path, p)

    def unlink(self, path):
        parent, filename = self.splitpath(path)
        p = self.getmeta(parent)
        p[’files’].remove(filename)
        self.putmeta(parent, p)
        p = self.getmeta(path)
        self.removemeta(path)
        blks = range((p[’st_size’] - 1)//bsize + 1)
        self.removedata(path, blks)

    def utimens(self, path, times = None):
        now = time()
        atime, mtime = times if times else (now, now)
        p = self.getmeta(path)
        p[’st_atime’] = atime
        p[’st_mtime’] = mtime
        self.putmeta(path, p)

    def write(self, path, data, offset, fh):
       p = self.getmeta(path)
        currblks = range((p[’st_size’] - 1)//bsize + 1)
        if offset > p[’st_size’]:
            fulldata = [(self.getdata(path, [i])[0] if i in currblks else
'').ljust(bsize, '\x00') for i in range(offset//bsize)] \
                    + [(self.getdata(path, [offset//bsize])[0][:offset % bsize]
if offset//bsize in currblks else '').ljust(offset % bsize, '\x00')]
            self.putdata(path, range(0, offset//bsize), fulldata)
        size = len(data)
        sdata = [data[:bsize - (offset % bsize)]] + [data[i:i+bsize] for i in
range(bsize - (offset % bsize), size, bsize)]
        blks = range(offset//bsize, (offset + size - 1)//bsize + 1)
        mod = blks[:]
        mod[0] = (self.getdata(path, [blks[0]])[0][:offset % bsize] if blks[0] in
currblks else '').ljust(offset % bsize, '\x00') + sdata[0]
        if len(mod[0]) != bsize and blks[0] in currblks:
            mod[0] = mod[0] + self.getdata(path, [blks[0]])[0][len(mod[0]):]
        mod[1:-1] = sdata[1:-1]
        if len(blks) > 1:
            mod[-1] = sdata[-1] + (self.getdata(path, [blks[-1]])[0][len(sdata[-
1]):] if blks[-1] in currblks else '')
        self.putdata(path, blks, mod)
        p['st_size']= offset + size if offset + size > p['st_size'] else
p['st_size']
        self.putmeta(path, p)
        return size

if __name__ == '__main__':
    if len(argv) != 4:
        print('usage: %s <mountpoint> <metaport> <dataport>' % argv[0])
        exit(1)
    logging.basicConfig(level=logging.DEBUG)
    fuse = FUSE(Memory(argv[2], argv[3]), argv[1], foreground=True, debug=True)
```