IMPLEMENATION.

Here I have used the metaserver and the dataserver to store the metadata and the data of the file system respectively.

I have modified the multilevelblockFS.py to implement this functionality.

I have also modified the given simpleHT.py code to create the metaserver and dataserver.

For every functionality we are getting metadata and data from the server, working on them and storing it back on the respective servers.

TEST CASES.

1. Created a file inside root directory. Checked the output through ls command and GUI.
2. Created a directory inside the root directory. Checked the output through the ls command and GUI.
3. Created a file inside a directory. Checked the output through the ls command and GUI.
4. Created a directory inside a directory. Checked the output through the ls command and GUI.
5. Truncated the data inside a file in the root directory. Checked the output.
6. Renamed the directory inside the root directory. Checked the output through ls command and GUI.
7. Renamed the directory inside the directory. Checked the output through ls command and GUI.
8. Removed the file inside the root directory. Checked the output through ls command and GUI.
9. Removed the directory inside the root directory. Checked the output through ls command and GUI.
10. Removed the directory inside the directory. Checked the output through ls command and GUI.
11. Checked the unlink function.

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

n = 8

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 = {}                                 # initialize self.files dictionary
        self.data = defaultdict(list)                           # initialize self.data as a default dictionary of list
        self.fd = 0
    self.n=8
    self.MetaServerPort = MetaServerPort
        self.DataServerPort = DataServerPort
    self.MetaServerHandle = xmlrpclib.ServerProxy('http://localhost:' + str(self.MetaServerPort) + '/')
```

```
        self.DataServerHandles = []
        for i in range(0,len(self.DataServerPort)):
            self.DataServerHandles.append(xmlrpclib.ServerProxy('http://localhost:' + str(self.DataServerPort[i]) + '/'))
        now = time()
        self.MetaServerHandle.put('/',pickle.dumps(dict(st_mode=(S_IFDIR | 0o755), st_ctime=now,st_mtime=now, st_atime=now,
st_nlink=2, files = [])))

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

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

    def create(self, path, mode):
    print('----------------------------------------------------------CREATE-----------------------------------------------------------')
        self.MetaServerHandle.put(path,pickle.dumps(dict(st_mode=(S_IFREG | mode), st_nlink=1,
                            st_size=0, st_ctime=time(), st_mtime=time(),
                            st_atime=time(),files=[],blocks=[])))

    parentpath,childpath = self.splitdata(path)                         # split the parent path and child path
    metaData = pickle.loads(self.MetaServerHandle.get(parentpath))
    metaData['files'].append(childpath)                                 # add the child path to parent file's metadata
    self.MetaServerHandle.put(parentpath,pickle.dumps(metaData))
        self.fd += 1
        return self.fd

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

    def getxattr(self, path, name, position=0):
        if self.MetaServerHandle.get(path) == -1:
            return ''                                       # Should return ENOATTR
        metaData = pickle.loads(self.MetaServerHandle.get(path))
        attrs = metaData.get('attrs', {})

        try:
            return attrs[name]
        except KeyError:
            return ''                                       # Should return ENOATTR

    def listxattr(self, path):
        if self.MetaServerHandle.get(path) == -1:
            return ''   # Should return ENOATTR
        metaData = pickle.loads(self.MetaServerHandle.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=[])
    parentpath,childpath = self.splitdata(path)                         # split the path into child path and parent path
```

```
self.MetaServerHandle.put(path,pickle.dumps(metaData))
metaData = pickle.loads(self.MetaServerHandle.get(parentpath))
metaData['st_nlink'] += 1                       # increment the st_nlink of the parent path by 1
metaData['files'].append(childpath)                     # add the child path to the files of the parent path
self.MetaServerHandle.put(parentpath,pickle.dumps(metaData))

def splitdata(self,path):
childpath = path[path.rfind('/')+1:]                    # storing the child path
parentpath = path[:path.rfind('/')]                     # storing the parent path
if parentpath == '':
    parentpath = '/'                            # default value for parent path
return parentpath,childpath                             # returning parent path and child path

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

def read(self, path, size, offset, fh):
metaData = pickle.loads(self.MetaServerHandle.get(path))
    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.DataServerHandles[blocks[i]].get(path + str(i))
    return result

def readdir(self, path, fh):
metadata=pickle.loads(self.MetaServerHandle.get(path))
    return ['.', '..'] + [x for x in metadata['files']]

def readlink(self, path):
    metaData = pickle.loads(self.MetaServerHandle.get(path))
    Data = self.readData(path,metaData[path]['blocks'])
    return Data

def removexattr(self, path, name):
    attrs = self.files[path].get('attrs', {})

    try:
        del attrs[name]
    except KeyError:
        pass                                    # Should return ENOATTR

def rename(self, old, new):
print('-----------------------------------------------------------RENAME-----------------------------------------')
oparentpath,ochildpath = self.splitdata(old)                    # split old address into old parentpath and old childpath
nparentpath,nchildpath = self.splitdata(new)                     # split new address into new parentpath and new childpath
metaData = pickle.loads(self.MetaServerHandle.get(old))
if metaData['st_mode'] & 0770000 == S_IFDIR:                        # check if the object to be moved is a file or a folder

    self.mkdir(new,S_IFDIR)
    filelist = metaData['files']
    for s in filelist:
        self.rename(old + '/' + s, new + '/' + s)
    self.rmdir(old)
else:
    self.create(new,33188)                              # call create function to create a new file in the new path
    self.files[nparentpath]['st_size'] = self.files[oparentpath]['st_size']          # copy size from the old parent path to the new
```

```
parent path
        self.data[nchildpath]= self.data[ochildpath]                    # copy the metadata of the old child path into the new child path
        self.files[oparentpath]['files'].remove(ochildpath)               # remove old child path from the old parent path

    def rmdir(self, path):
    print('-----------------------------------------------------RMDIR-----------------------------------------------')
    metaData=self.MetaServerHandle.pop_entry(path)
        parentpath,childpath = self.splitdata(path)                       # split the path into parent path and new path
    metaData = pickle.loads(self.MetaServerHandle.get(parentpath))
    metaData['files'].remove(childpath)                                 # remove the files related to the childpath from the parent path
        metaData['st_nlink'] -= 1                                   # decrement the st_nlink by 1
    self.MetaServerHandle.put(parentpath,pickle.dumps(metaData))


    def setxattr(self, path, name, value, options, position=0):
        # Ignore options
        if self.MetaServerHandle.get(path) == -1:
            return ''                                       # Should return ENOATTR
        metaData = pickle.loads(self.MetaServerHandle.get(path))
        attrs = metaData.setdefault('attrs', {})
        attrs[name] = value
        metaData.set('attrs', attrs)
        self.MetaServerHandle.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):
    print('-------------------------------------------SYMLINK-----------------------------------------------------')
    print('source '+str(source))
    print('target '+str(target))
    self.files[target] = dict(st_mode=(S_IFLINK | 0o777), st_nlink=1, st_size=len(source))
    d1 = target[target.rfind('/')+1:]                                # find the last occurence of '/' in the path and add 1 to it to do slicing
    self.data[target] = [source[i:i+n] for i in range(0, len(source), n)]               # copying the data from the source to target, 8 bits at
a time


    def truncate(self, path, length, fh=None):
    data = []
    metaData = pickle.loads(self.MetaServerHandle.get(path))
    blocks = metaData['blocks']
    trunblklen = length // n
    trunstrlen = length % n
    for i in range (0, len(blocks)):
        ndata = self.DataServerHandles[blocks[i]].get(path + str(i))
        data += [''.join(ndata)]
    if length < metaData['st_size']:
        newData = data[:trunblklen] + [''.join(data[trunblklen][:trunblklen])]
    offset = metaData['st_size']
    metaData['st_size'] = length
    metaData['blocks'] = blocks
    for i in range(0,len(newData)):
        self.DataServerHandles[blocks[i]].put(path + str(i),newData[i])
    self.MetaServerHandle.put(path,pickle.dumps(metaData))


    def unlink(self, path):
    parentpath,childpath=self.splitdata(path)                           # seperate parentpath and childpath
    metaData = pickle.loads(self.MetaServerHandle.get(parentpath))                  # bring metadata from the metaserver to the
client
    metaData['files'].remove(childpath)                             # remove childpath from parentpath's metadata
```

```
    self.MetaServerHandle.put(parentpath,pickle.dumps(metaData))                # send updated metadata back to the
metaserver

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

    def write(self, path, data, offset, fh):
        newDataInBlocks = []
        blocks = []
        x = hash(path)
        metaData = pickle.loads(self.MetaServerHandle.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):]
        j = 1
        for i in range(0,len(newData),self.n):
            newDataInBlocks.append(newData[i : i + self.n])
            blocks.append((x + j - 1) % len(self.DataServerPort))
            j += 1;

        self.writeData(path,newDataInBlocks,blocks)
        metaData['st_size'] = len(newData)
        metaData['blocks'] = blocks
        self.MetaServerHandle.put(path,pickle.dumps(metaData))
        return len(data)

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




if __name__ == '__main__':
    if len(argv) < 4:
        print('usage: %s <mountpoint> <metaserver port> <dataserver port1> ...' % 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)
```