**Distributed File System with Python**

In this assignment, we will use the Sanketplus/PyDFS library in Github (see link below) to make a distributed file system application. Be sure to review link below to become familiar with this code.

https://github.com/sanketplus/PyDFS/tree/srecon

You will first need to create a minion node (see code below). Use this code to create two different minion nodes. Be sure to review the lecture video in Teams as well.

**# https://github.com/sanketplus/PyDFS/blob/srecon/pydfs/minion.py**

**import rpyc**

**import os**

**import sys**

**import logging**

**from rpyc.utils.server import ThreadedServer**

**DATA\_DIR = "/tmp/minion/"**

**PORT = 8888**

**logging.basicConfig(level=logging.DEBUG)**

**class Minion(rpyc.Service):**

**def exposed\_put(self, block\_id, data, minions):**

**logging.debug("put block: " + block\_id)**

**out\_path = os.path.join(DATA\_DIR, block\_id)**

**with open(out\_path, 'w') as f:**

**f.write(data)**

**if len(minions) > 0:**

**self.forward(block\_id, data, minions)**

**def exposed\_get(self, block\_id):**

**logging.debug("get block: " + block\_id)**

**block\_addr = os.path.join(DATA\_DIR, block\_id)**

**if not os.path.isfile(block\_addr):**

**logging.debug("block not found")**

**return None**

**with open(block\_addr) as f:**

**return f.read()**

**def forward(self, block\_id, data, minions):**

**logging.debug("forwarding block: " + block\_id + str(minions))**

**next\_minion = minions[0]**

**minions = minions[1:]**

**host, port = next\_minion**

**rpyc.connect(host, port=port).root.put(block\_id, data, minions)**

**if \_\_name\_\_ == "\_\_main\_\_":**

**PORT = int(sys.argv[1])**

**DATA\_DIR = sys.argv[2]**

**if not os.path.isdir(DATA\_DIR):**

**os.mkdir(DATA\_DIR)**

**logging.debug("starting minion")**

**rpyc\_logger = logging.getLogger('rpyc')**

**rpyc\_logger.setLevel(logging.WARN)**

**t = ThreadedServer(Minion(), port=PORT, logger=rpyc\_logger, protocol\_config={**

**'allow\_public\_attrs': True,**

**})**

**t.start()**

Next, you will need to start the master node.

**# https://github.com/sanketplus/PyDFS/blob/srecon/pydfs/master.py**

**import rpyc**

**import uuid**

**import math**

**import random**

**import configparser**

**import signal**

**import pickle**

**import sys**

**import os**

**from rpyc.utils.server import ThreadedServer**

**BLOCK\_SIZE = 100**

**REPLICATION\_FACTOR = 2**

**MINIONS = {"1": ("127.0.0.1", 8000),**

**"2": ("127.0.0.1", 9000),}**

**class MasterService(rpyc.Service):**

**"""**

**file\_block = {'file.txt': ["block1", "block2"]}**

**block\_minion = {"block1": [1,3]}**

**minions = {"1": (127.0.0.1,8000), "3": (127.0.0.1,9000)}**

**"""**

**file\_block = {}**

**block\_minion = {}**

**minions = MINIONS**

**block\_size = BLOCK\_SIZE**

**replication\_factor = REPLICATION\_FACTOR**

**def exposed\_read(self, file):**

**mapping = []**

**# iterate over all of file's blocks**

**for blk in self.file\_block[file]:**

**minion\_addr = []**

**# get all minions that contain that block**

**for m\_id in self.block\_minion[blk]:**

**minion\_addr.append(self.minions[m\_id])**

**mapping.append({"block\_id": blk, "block\_addr": minion\_addr})**

**return mapping**

**def exposed\_write(self, file, size):**

**self.file\_block[file] = []**

**num\_blocks = int(math.ceil(float(size) / self.block\_size))**

**return self.alloc\_blocks(file, num\_blocks)**

**def alloc\_blocks(self, file, num\_blocks):**

**return\_blocks = []**

**for i in range(0, num\_blocks):**

**block\_id = str(uuid.uuid1()) # generate a block**

**minion\_ids = random.sample( # allocate REPLICATION\_FACTOR number of minions**

**list(self.minions.keys()), self.replication\_factor)**

**minion\_addr = [self.minions[m] for m in minion\_ids]**

**self.block\_minion[block\_id] = minion\_ids**

**self.file\_block[file].append(block\_id)**

**return\_blocks.append(**

**{"block\_id": block\_id, "block\_addr": minion\_addr})**

**return return\_blocks**

**if \_\_name\_\_ == "\_\_main\_\_":**

**t = ThreadedServer(MasterService(), port=2131, protocol\_config={**

**'allow\_public\_attrs': True,**

**})**

**t.start()**

Finally, the follow code will allow you to query the distributed file system (see below).

**# https://github.com/sanketplus/PyDFS/blob/srecon/pydfs/client.py**

**import rpyc**

**import sys**

**import os**

**import logging**

**logging.basicConfig(level=logging.DEBUG)**

**def get(master, file):**

**file\_table = master.read(file)**

**if not file\_table:**

**logging.info("file not found")**

**return**

**for block in file\_table:**

**for host, port in block['block\_addr']:**

**try:**

**con = rpyc.connect(host, port=port).root**

**data = con.get(block['block\_id'])**

**if data:**

**sys.stdout.write(data)**

**break**

**except Exception as e:**

**continue**

**else:**

**logging.error("No blocks found. Possibly a corrupt file")**

**def put(master, source, dest):**

**size = os.path.getsize(source)**

**blocks = master.write(dest, size)**

**with open(source) as f:**

**for block in blocks:**

**data = f.read(master.block\_size)**

**block\_id = block['block\_id']**

**minions = block['block\_addr']**

**minion = minions[0]**

**minions = minions[1:]**

**host, port = minion**

**con = rpyc.connect(host, port=port)**

**con.root.put(block\_id, data, minions)**

**def main(args):**

**con = rpyc.connect("localhost", port=2131)**

**master = con.root**

**if args[0] == "get":**

**get(master, args[1])**

**elif args[0] == "put":**

**put(master, args[1], args[2])**

**else:**

**logging.error("try 'put srcFile destFile OR get file'")**

**if \_\_name\_\_ == "\_\_main\_\_":**

**main(sys.argv[1:])**

One of the problems with this code from Github is that it won’t save state between stopping and starting the Master file. For this assignment, you will need to implement that functionality. Use the two files in the zip file in Canvas to track the internal state (“PyDFS\_Master\_File\_Block\_File.dat” and “PyDFS\_Master\_Block\_Minion\_File.dat”). The first file will hold the dictionary contents of the file\_block dictionary in the Master program. The second file will hold the dictionary contents of the block\_minion dictionary in the Master program. Both are needed to implement the desired functionality.

(Hint: the only file you need to modify to implement this functionality is the Master program. Also, pay close attention to what the alloc\_blocks function is doing in the Master file. This will be key. You may also want to use the “ast” library in Python to assist with reading a file into a dictionary.)

**\*\*\*\*Be sure to include screenshots of your working code in addition to submitting the Python code itself.**

**Also, you must implement this functionality using the two files I mentioned previously (e.g., “PyDFS\_Master\_File\_Block\_File.dat” and “PyDFS\_Master\_Block\_Minion\_File.dat”) in order to receive credit for this assignment.**

Submit your Python source files and any other relevant material to Canvas.