Assignment 04: Distributed File systems
University of Puerto Rico at Rio Piedras
Department of Computer Science
CCOM4017: Operating System

Student: Joel Maldonado Rivera
Student ID: 801-14-3804

Student Discussion: Peers who at one point or another I discussed part of the project with to clear doubts or to help them.

1. Heriberto Camacho = discussed on how to run the commands on the command line.
2. John Wilson = discussed on how to divide the file, running the project in the command line and when I had a bug we would suggest what to check.
3. Roberto Lopez = showed him how to run the commands on the command line.

Outside References: External resources that helped me understand and deal with the project.

1. https://stackoverflow.com/questions/14899506/displaying-better-error-message-than-no-json-object-could-be-decoded
2. https://stackoverflow.com/questions/22547822/error-on-django-runserver-overflowerror-getsockaddrarg-port-must-be-0-65535
3. https://stackoverflow.com/questions/15728564/how-to-open-read-and-write-files-in-python-2-7-converting-code-from-fortran
4. https://stackoverflow.com/questions/5161167/python-handling-specific-error-codes
5. https://docs.python.org/2/library/socketserver.html
6. https://realpython.com/python-sockets/
7. http://www.java2s.com/Code/Python/Network/ErrorHandlingExample.htm
8. https://www.geeksforgeeks.org/socket-programming-python/
9. https://docs.python.org/2/library/uuid.html

INTRODUCTION:

In this project the student will implement the main components of a file system by implementing a simple, yet functional, distributed file system (DFS).The project will expand the student knowledge of the components of file systems (inodes, and data blocks), will develop the student skills in inter process communication, and will increase system security awareness on the students.

The components to implement are:
- A metadata server, which will function as a inodes repository
- Data servers, that will serve as the disk space for file data blocks
- list client, that will list the files available in the DFS

- copy client, that will copy files from and to the DFS

FUNCTIONS USED:

Functions:

General Functions
* getEncodedPacket(): returns a seriliazed packet ready to send through the network.First you need to build the packets.See BuildXPacket functions.
* DecodePacket(packet): Receives a serialized message and turns it into a packet object.
* getCommand(): Returns the command type of the packet

Packet Registration Functions
* BuildRegPacket(addr, port): Builds a registration packet.
* getAddr():Returns the IP address of a server. Useful for registration packets
* getPort(): Returns the Port number of a server. Useful for registration packets

Packet List Functions
* BuildListPacket(): Builds a list packet for file listing
* BuildListResponse(filelist): Builds a list response packet
* getFileArray(): Returns a list of files

Get Packet Functions
* BuildGetPacket(fname): Builds a get packet to get fname.
* BuildGetResponse(metalist, fsize): Builds a list of data node servers with the blocks of a file, and file size.
* getFileName(): Returns the file name in a packet.
* getDataNodes(): Returns a list of data servers.

Put Packet Functions
* BuildPutPacket(fname, size): Builds a put packet to put fname and file size.
* getFileInfo(): Returns the file info in a packet.
* BuildPutResponse(metalist): Builds a list of data node servers where a file data blocks can be stored. I.E a list of available data servers.
* BuildDataBlockPacket(fname, block_list): Builds a data block packet. Contains the file name and the list of blocks for the file.
* getDataBlocks():
* Returns a list of data blocks

Get Data block Functions
* BuildGetDataBlockPacket(blockid): Builds a get data block packet. Useful when requesting a data block to a data node.
* getBlockID(): Returns a the block_id from a packet.

FILES:

The meta data server

The meta data server contains the metadata (inode) information of the files in your file system. It will also keep registry of the data servers that are connected to the DFS.

Your metadata server must provide the following services:

1. Listen to the data nodes that are part of the DFS. Every time a new data node registers to the DFS the metadata server must keep the contact information of that data node. This is (IP Address, Listening Port).
   * To ease the implementation the DFS the directory file system must contain three things:
     - the path of the file in the file system
     - the nodes that contain the data blocks of the files
     - the file size

2. Every time a client (commands list or copy) contacts the meta data server for:
   * requesting to read a file: the metadata server must check if the file is in the DFS database, and if it is, it must return the nodes with its blocks_ids that contain the file.
   * requesting to write a file: the metadata server must:
     - insert in the database the path of the new file and name of the file, and its size.
     - return a list of available data nodes where to write the chunks of the file
     - then store the data blocks that have the information of the data nodes and the block ids of the file.
   * requesting to list files:
     - the metadata server must return a list with the files in the DFS and their size.

The metadata server must be run:

```
python meta-data.py <port, default=8000>
```

If no port is specified the port 8000 will be used by default.

The data node server

The data node is the process that receives and saves the data blocks of the files. It must first register with the metadata server as soon as it starts its execution. The data node receives the data from the clients when the client wants to write a file, and returns the data when the client wants to read a file.

Your data node must provide the following services:

1. Listen to writes (puts):

    * The data node will receive blocks of data, store them using and unique id, and return the unique id.
    * Each node must have its own blocks storage path. You may run more than one data node per system.
2. Listen to reads (gets)
    * The data node will receive request for data blocks, and it must read the data block, and return its content.

The data nodes must be run like so:
`python data-node.py <server address> <port> <data path> <metadata port,default=8000>`

Server address is the meta data server address, port is the data-node port number, data path is a path to a directory to store the data blocks, and metadata port is the optional metadata port if it was ran in a different port other than the default port.

Note: Since you most probably do not have many different computers at your disposition, you may run more than one data-node in the same computer but the listening port and their data block directory must be different.

The list client

The list client just sends a list request to the meta data server and then waits for a list of file names with their size.

The output will look like:
```
/home/cheo/asig.cpp 30 bytes
/home/hola.txt 200 bytes
/home/saludos.dat 2000 bytes
```

The list client cant be run like so:

python ls.py ,server,port, default=8000

Where server is the metadata server IP and port is the metadata server port. If the default port is not indicated the default port is 8000 and no ':' character is necessary.

The copy client

The copy client is more complicated than the list client. It is in charge of copying the files from and to the DFS.

The copy client will do the following:

1. Write files in the DFS
    * The client must send to the metadata server the file name and size of the file to write.
    * Wait for the metadata server response with the list of available data nodes.

    * Send the data blocks to each data node.
        - this is done by dividing by the nodes and using mod in order to send leftover data as well
2. Read files from the DFS
    * Contact the metadata server with the file name to read.
    * Wait for the block list with the bloc id and data server information
    * Retrieve the file blocks from the data servers.
        - This part will depend on the division algorithm used in step 1.

The copy client must be run:

Copy from DFS:
`python copy.py <server>:<port>:<dfs file path> <destination file>`

To DFS:
`python copy.py <source file> <server>:<port>:<dfs file path>`

Where server is the metadata server IP address, and port is the metadata server port.

Creating an empty database
The script createdb.py generates an empty database *dfs.db* for the project.

`python createdb.py`

File Execution order:

1.createdb.py
2.meta-data.py
3.data-node.py
4.copy.py(copy to)
5.ls.py
6.copy.py(copy from)
7.Done!