Contents

Raymonds Algorithm - A Distributed Approach

Sabbir Rashid
Renssselaer Polytechnic Institute

I INTRODUCTION

In this report, an implementation of Raymond's Algorithm is discussed, where the design for file operations, Client and Server Socketing, and Raymond's functionality are described.

I.I Problem Statement

The goal for this project was to implement a distributed application were files could be created, appended to, read, and deleted. In order to ensure that no two processes could access a single file at once, Raymond's Algorithm can be used to ensure mutual exclusion.

II METHODOLOGY

The goal of this project is to implement Raymond's Algorithm in the a distributed environment, such as Amazon Web Services EC2 Instances. In order to do so, we had to create a Process class and methods for creating Processes, Client and MultiThreaded Server Classes, and a class to implement Raymond's Algorithm Functionality.

In order to implement Raymond's Algorithm, we used the Java Buffered and File Reader and Writer packages, as well as other standard packages. In order to implement the server and client, we used the socket and server socket packages.

II.I Tree Implementation

In order to determine which processes had which neighbors based on an input tree file, the following tree class was created, with the CreateTree Method. This method creates an Array List of Processes with Neighbors defined by the input file. When a tree is created, first, all of the given processes are added to an ArrayList. Then, once they are added, each process is checked again to see what neighbors they have and their neighbors are added to a neighbors ArrayList for each process.

```java
package raymonds;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

public class Tree {

        public static ArrayList<Process> CreateTree(String inputFile) throws IOException
        {
```

```
ArrayList<Process> processes = new ArrayList<Process>();
FileReader fr = new FileReader(inputFile);
BufferedReader br = new BufferedReader(fr);
String input = br.readLine();
boolean first = true;
while(input!=null)
{
        if ( first )
        {
                processes.add(new Process(input.substring(1, 2),Process.HolderEnum.
                    Neighbor,false,false));
                processes.get(0).addNeighbor(new Process(input.substring(3, 4),Process.
                    HolderEnum.Neighbor,false,false));
                processes.add(new Process(input.substring(3, 4),Process.HolderEnum.
                    Neighbor,false,false));
                processes.get(1).addNeighbor(new Process(input.substring(1, 2),Process.
                    HolderEnum.Neighbor,false,false));
                first = false;
        }
        else
        {
                int index = 0;
                boolean found=false;
                for(int i=0;i<processes.size();i++)
                {
                        if(input.substring(1,2).equals(processes.get(i).getProcessID()))
                        {
                                found=true;
                                index=i;
                        }

                }
                if (!found)
                {
                        processes.add(new Process(input.substring(1, 2),Process.
                            HolderEnum.Neighbor,false,false));
                        processes.get(processes.size()-1).addNeighbor(new Process(
                            input.substring(3, 4),Process.HolderEnum.Neighbor,false,
                            false));
                }
                else
                {
                        if (!processes.get(index).getNeighbors().contains(new Process(
                            input.substring(3, 4),Process.HolderEnum.Neighbor,false,
                            false)))
                        {
                                processes.get(index).addNeighbor(new Process(input.
                                    substring(3, 4),Process.HolderEnum.Neighbor,false,
                                    false));
                                //System.out.println("CONTAINS 1");
                            }
                        }
                        found=false;
                        index = 0;
                        for(int i=0;i<processes.size();i++)
                        {
                            if(input.substring(3,4).equals(processes.get(i).getProcessID()))
                            {
                                found=true;
                                index=i;
                            }
                        }
                        if (!found)
                        {
                            processes.add(new Process(input.substring(3, 4),Process.
                                HolderEnum.Neighbor,false,false));
                            processes.get(processes.size()-1).addNeighbor(new Process(
                                input.substring(1, 2),Process.HolderEnum.Neighbor,false,
                                false));
                        }
                        else
                        {
                            if (!processes.get(index).getNeighbors().contains(new Process(
                                input.substring(1, 2),Process.HolderEnum.Neighbor,false,
                                false)))
                            {
                                processes.get(index).addNeighbor(new Process(input.
                                    substring(1, 2),Process.HolderEnum.Neighbor,false,
                                    false));
                                //System.out.println("CONTAINS 2");
                            }
                        }
                    }
                    input=br.readLine();
                }
                br.close();
                return processes;
            }

}
```

II.II Raymond's Functions

Our java implementation of Raymond's Algorithm functionality is listed and described in this section.

II.II.i assignToken(Process p)

First, we can assign the token by inputing a process and checking all of the usual parameters of Raymond's algorithm to see if we are able to assign the token to the process.

```
public static void assignToken(Process p, Process holderProc) {
        if ( (p.holderEnum == Process.HolderEnum.Self) && (!p.usingResource) && (!p.
            requestQueue.isEmpty()) ) {
                holderProc = p.requestQueue.pop() ;

                if (p.getProcessID() == holderProc.getProcessID()) { //i.e. the process p is at
                    the front of its own queue
                        p.holderEnum = Process.HolderEnum.Self;
                } else {
                        p.holderEnum = Process.HolderEnum.Neighbor;
                        holderProc.holderEnum = Process.HolderEnum.Self ;
                }

                p.asked = false;

                if (p.holderEnum == Process.HolderEnum.Self) {
                        p.usingResource = true;
                } else {
                        assignToken(p, holderProc); // Check this, supposed to be "send token
                            to holder"
                }
        }
}
```

We can send a request for the token by inputing the process that is requesting the token and adding the request to the processe's request queue

II.II.ii sendRequest(Process p)

```
public static void sendRequest(Process p, Process holderProc) {
        if ( (p.holderEnum != Process.HolderEnum.Self) && (!p.requestQueue.isEmpty()) && (!
            p.asked) ) {
                sendRequest(p, holderProc);
                p.asked = true;
        }
}
```

II.II.iii requestResource(Process p)

Calls of the necessary functions in order to request the token.

```
public static void requestResource(Process p, Process holderProc) {
        p.requestQueue.push(p);
        assignToken(p, holderProc);
        sendRequest(p, holderProc);
}
```

II.II.iv releaseResource(Process p)

Once the process is done using the resource, it releases it from being used.

```
    public static void releaseResource(Process p, Process holderProc) {
        p.usingResource = false;
        assignToken(p, holderProc);
        sendRequest(p, holderProc);
    }
```

II.II.v receivedRequestFromNeighbor(Process p, Process neighbor)

This function is used in a similar fashion to requestResource, except it does it for the neighbor of the process

```
    public static void receivedRequestFromNeighbor(Process p, Process holderProc, Process neighbor
      ) {
        p.requestQueue.push(neighbor);
        assignToken(p, holderProc);
        sendRequest(p, holderProc);
    }
```

II.II.vi receivedToken(Process p)

Assigns the token to the given function when it receives it

```
    public static void receivedToken(Process p, Process holderProc) {
        p.holderEnum = Process.HolderEnum.Self ;
        holderProc = p;
        assignToken(p, holderProc);
        sendRequest(p, holderProc);
    }
```

II.III Client Implementation

The client was implemented using the java Net and IO packages. The client is able to decide which file operation to conduct. Some input checking is done to make sure that the client exists in the process tree.

```
package distributed;

import java.io.*;
import java.net.*;
import java.util.ArrayList;

import raymonds.Process;
import raymonds.Tree;


public class Client {
    public static Process clientProcess = new Process();

    public static void main(String[] args) throws IOException {
    if (args.length != 3) {
        System.err.println(
            "Usage: java Client <host name> <port number> <client process id>");
        System.exit(1);
    }

    String hostName = args[0];
    int portNumber = Integer.parseInt(args[1]);
    String clientID=args[2];
    try {
                    boolean found = false;
                    ArrayList<Process> processes = Tree.CreateTree("tree.txt");
                    for (Process p : processes) {
                            //System.out.println(p.getProcessID());
                            if (Integer.parseInt(clientID) == Integer.parseInt(p.getProcessID())){
                                    found = true;
                                    System.out.println("Found the Client Process in the tree ... ");
                                    clientProcess = p;
                            }
                    }
                    if (!found){
                            System.err.println("Error: Client Process " + clientID + " entered does
                                not exist in tree");
                            System.exit(1);
                    }
            } catch (Exception e){
                    System.err.println("Error: " + e);
            }

    System.out.println("CLIENT: About to try to create Client Socket");
    try (
        Socket clientSocket = new Socket(hostName, portNumber);
//              System.out.println("CLIENT: Created Client Socket");
            PrintWriter out =
            new PrintWriter(clientSocket.getOutputStream(), true);
//              System.out.println("CLIENT: Initiated print writer output stream.");
/*              ObjectOutputStream oos =
                    new ObjectOutputStream(clientSocket.getOutputStream());*/
            BufferedReader in =
            new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));
//              System.out.println("CLIENT: Initiated print buffered reader input stream.");
        BufferedReader stdIn =
            new BufferedReader(
                new InputStreamReader(System.in));
//              System.out.println("CLIENT: Initiated print buffered reader stdIn.");
            ){

/*              Process clientProcess = new Process();
```

```
                    ArrayList<Process> processes = Tree.CreateTree("tree.txt");
                            for (Process p : processes) {
                                    if (processID == p.getProcessID()){
                                            clientProcess = p;
                                    }
                            }
  */
                    out.println(clientID);
  /*                oos.writeObject(clientProcess);
                    oos.flush();*/
                    while(true){
                    String newLine;
                    while ( true ) {
                            newLine = in.readLine();
                            if (newLine.equalsIgnoreCase("ACQUIRETOKEN")){
                                    for (Process n : clientProcess.neighbors){
                                            Raymonds.requestResource(clientProcess, n);
                                    }
                                    break;
                            }
                            else if (newLine.equalsIgnoreCase("END"))
                            {
                                    break;
                            }
                            System.out.println(newLine);
                    }
                    String userInput;
                    System.out.println("CLIENT: About to wait for user input.");
                    /*                  System.out.println("1: create <filename>: creates an empty file named
                         <filename>");
                    System.out.println("2: delete <filename>: deletes file named <filename>");
                    System.out.println("3: read <filename>: displays the contents of <filename>");
                    System.out.println("4: append <filename> <line>: appends a <line> to <filename>");
                    System.out.println("5: exit: exits the program");
                     */
                    userInput = stdIn.readLine();
                    out.println(userInput);
/*                  while ((userInput = stdIn.readLine()) != null) {
                            System.out.println("In stdIn while loop");
                            out.println(userInput);
                            out.flush();

                            while ( true ) {
                            newLine = in.readLine();
                            if (newLine.equalsIgnoreCase("END"))
                            {
                                break;
                            }
                            System.out.println(newLine);
```

```
                }
                }*/
                }
            //System.out.println("CLIENT: Exited stdIn while loop.");
            //clientSocket.close();
        } catch (UnknownHostException e) {
            System.err.println("Don't know about host " + hostName);
            System.exit(1);
        } catch (IOException e) {
            System.err.println("Couldn't get I/O for the connection to " +
                hostName);
            System.exit(1);
        }
    }
}
```

II.IV Single Threaded Server

The Single Threaded Server implementation is listed below. This class was the initial attempt at creating a server, and provided a sandbox to learn how a server handled a single client instance.

```
package sockets;

import java.io.*;
import java.net.*;

import main.Main;

public class Server {
    public static void main(String[] args) throws IOException {

        if (args.length != 1) {
            System.err.println("Usage: java Server <port number>");
            System.exit(1);
        }

        int portNumber = Integer.parseInt(args[0]);
        System.out.println("SERVER: About to try to create a server socket.");
        try {
                System.out.println("SERVER: Creating server socket.");
            ServerSocket serverSocket =
                new ServerSocket(Integer.parseInt(args[0]));
            System.out.println("SERVER: About to set Client Socket.");
            Socket clientSocket = serverSocket.accept();
            System.out.println("SERVER: Created Client Socket.");
            PrintWriter out =
                new PrintWriter(clientSocket.getOutputStream(), true);
            System.out.println("SERVER: Created print writer out.");
            BufferedReader in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));
            System.out.println("SERVER: Created buffered reader.");

            System.out.println("SERVER: In try. About to enter while loop.");

        /*      out.println("Select  the  following  command that you want to execute:");
            out.flush();
                out.println("1:  create  <filename>: creates an empty file  named <filename>");
                out.flush();
                out.println("2:  delete  <filename>: deletes  file  named <filename>");
                out.flush();
                out.println("3:  read  <filename>: displays the contents  of  <filename>");
                out.flush();
                out.println("4:  append <filename> <line>: appends a <line> to <filename>");
         */
            //while ((inputLine = in.readLine()) != null)  {
            int  i   = 0;
            while (i  < 100){
                        //String  result  = console.nextLine();
                        //String  result  = inputLine;
                String  result  = in.readLine();
                        //Note: Calling  create,  delete,  read,  and append go here:
                        File  testFile  = null;
                        if(result.substring(0,6).equalsIgnoreCase("create"))
                        {
                                out.println("Creating File ... ");
                                testFile  = Main.CreateFile(result.substring(7, result.length()));
                        }
                        else  if(result.substring(0,6).equalsIgnoreCase("delete"))
                        {
                                out.println("Deleting File ... ");
                                Main.DeleteFile(result.substring(7, result.length()));
                        }
                        else  if(result.substring(0,4).equalsIgnoreCase("read"))
                        {
                                String temp = Main.ReadFile(result.substring(5,result.length()));
                                out.println("Reading File...\n" + temp);
                                out.flush();
                        }
                        else  if(result.substring(0,6).equalsIgnoreCase("append"))
                        {
                                out.println("Appending to File...");
                                String tmp = result.substring(7, result.length());
                                int  index = tmp.indexOf(' ');
                                Main.AppendFile(tmp.substring(0,index),tmp.substring(index+1,tmp.
                                    length()));
                        }
                        else  if(result.substring(0,4).equalsIgnoreCase("exit"))
                        {
                                out.println("Exiting ... ");
```

```
                                out.flush();
                                Main.ExitConnection();
                        }
                        else
                                out.println("Error: Invalid Command");

            }
            System.out.println("SERVER: In try. Exited while loop.");
            serverSocket.close();
        } catch (IOException e) {
            System.out.println("Exception caught when trying to listen on port "
                + portNumber + " or listening for a connection");
            System.out.println(e.getMessage());
        }
    }
}
```

II.V Multithreaded Server

The multithreaded server implementation allows for multiple clients to access the server. The main functionality of the program is found in this class, as the server sends the clients possible tasks and reads and executes the commands. When initiating a server, this class checks that the server ID exists in the tree. When a client attempts to connect to a server, the server makes sure that the client is a direct neighbor of the server.

```
package distributed;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.Scanner;

import distributed.Main;
import raymonds.Process;
import raymonds.Tree;

public class MultiThread {
        public static Process serverProcess = new Process();
        public static LinkedHashMap<String,String> tokenMap = new LinkedHashMap<String,String
            >();
        public static LinkedHashMap<String,String> tokenOwner = new LinkedHashMap<String,String
            >();

        public static void main(String[] args) throws Exception {
```

```
        if (args.length != 2) {
            System.err.println("Usage: java MultiThread <port number> <server process id
                >");
            System.exit(1);
        }
        System.out.println("Running MultiThread...");
        int portNumber = Integer.parseInt(args[0]);
        String serverID = args[1];

        try {
            boolean found = false;
            ArrayList<Process> processes = Tree.CreateTree("tree.txt");
            for (Process p : processes) {
                //System.out.println(p.getProcessID());
                if (Integer.parseInt(serverID) == Integer.parseInt(p.getProcessID())){
                    found = true;
                    System.out.println("Found the Server Process in the tree ... ");
                    serverProcess = p;
                }
            }
            if (!found){
                System.err.println("Error: Server Process " + serverID + " entered does
                    not exist in tree");
                System.exit(1);
            }
        } catch (Exception e){
            System.err.println("Error: " + e);
        }
        int clientID = 1;

        @SuppressWarnings("resource")
        ServerSocket m_ServerSocket = new ServerSocket(portNumber);
        System.out.println("About to enter Wait For Client Loop...");
        while (true) {
            Socket clientSocket = m_ServerSocket.accept();
            ClientServiceThread cliThread = new ClientServiceThread(clientSocket, clientID);
            cliThread.start();
        }
    }

    public static void runMultiThread(String port, int clientID) throws Exception {
        System.out.println("Running MultiThread...");
        int portNumber = Integer.parseInt(port);

        @SuppressWarnings("resource")
        ServerSocket m_ServerSocket = new ServerSocket(portNumber);

        while (true) {
            Socket clientSocket = m_ServerSocket.accept();
```

```
                ClientServiceThread cliThread = new ClientServiceThread(clientSocket, clientID);
                cliThread.start();
            }
        }

        public static void runMultiThread(String port, String clientID) throws Exception {
            System.out.println("Running MultiThread...");
            int portNumber = Integer.parseInt(port);

            @SuppressWarnings("resource")
            ServerSocket m_ServerSocket = new ServerSocket(portNumber);

            while (true) {
                Socket clientSocket = m_ServerSocket.accept();
                ClientServiceThread cliThread = new ClientServiceThread(clientSocket, Integer.
                    parseInt(clientID));
                cliThread.start();
            }
        }
}

class ClientServiceThread extends Thread {
        Socket clientSocket;

        int clientID = -1;
        boolean running = true;


        ClientServiceThread(Socket s, int i) {
            clientSocket = s;
            clientID = i;
/*          try {
                ArrayList<Process> processes = Tree.CreateTree("tree.txt");
                for (Process p : processes) {
                    if (clientID == Integer.parseInt(p.getProcessID())){
                        clientProcess = p;
                    }
                }
                } catch (Exception e){
                    System.out.println("Error: " + e);
                }
                */
        }

        public void run() {
            //System.out.println("Accepted Client : ID - " + clientID + " : Address - "
            //              + clientSocket.getInetAddress().getHostName());
            try {
                BufferedReader  in = new BufferedReader(new InputStreamReader(clientSocket.
                    getInputStream()));
                System.out.println("SERVER: Created buffered reader in.");
                PrintWriter   out = new PrintWriter(new OutputStreamWriter(clientSocket.
                    getOutputStream()),true);
                System.out.println("SERVER: Created print writer out.");

                while (running) {
                        System.out.println("SERVER: In running loop.");
                        System.out.println("SERVER: In try. About to enter while loop.");
                        System.out.println("Reading Client ID");
                        String clientID = in.readLine();
                        boolean found = false;
                        for (Process p : MultiThread.serverProcess.getNeighbors()){
                                if (Integer.parseInt(clientID) == Integer.parseInt(p.
                                    getProcessID())){
                                        System.out.println("Found the client in the list of
                                            Neighbors...");
                                        found = true;
                                }
                        }
                        if (found){
                        System.out.println("Accepted Client : ID - " + clientID + " : Address
                            - "
                                        + clientSocket.getInetAddress().getHostName());
                        runSocketIO(out, in, clientID, MultiThread.tokenMap, MultiThread.
                            tokenOwner);
                        }
                        else {
                                System.out.println("Client : " + clientID + " : is not a
                                    neighbor of Server : " + MultiThread.serverProcess.
                                    getProcessID() + ":" );
                                clientSocket.close();
                        }

                }
        } catch (Exception e) {
                e.printStackTrace();
        }
}

public static void runSocketIO(PrintWriter out, BufferedReader in, String processID,
    LinkedHashMap<String,String> tokenMap, LinkedHashMap<String,String> tokenOwner)
    throws IOException {
        // TODO Auto-generated method stub

        while(true) {
                out.println("SERVER: Select the following command that you want to execute:")
                    ;
                    out.println("1: create <filename>: creates an empty file named <filename>");
                    out.println("2: delete <filename>: deletes file named <filename>");
                    out.println("3: read <filename>: displays the contents of <filename>");
                    out.println("4: append <filename> <line>: appends a <line> to <filename>");
                    out.println("5: list : lists token owner and contents");
                    out.println("6: exit: exit");
                    out.println("END");
                    if (true) {
                            String result = in.readLine();
                            //Note: Calling create, delete, read, list , and append go here:
                            // Ordered by string length
                            if(result.equals("5")) // list shortcut
                            {
                                    for (String token : tokenMap.keySet()){
                                            out.println("Token Name: \"" + token + "\"\nToken
                                                Owner: \"" + tokenOwner.get(token) + "\"\
                                                nContents: \n" + tokenMap.get(token) + "\n");
//                                          out.println("END");
                                    }
                            }
                            else if(result.equals("6")) // exit shortcut
                            {
                                    break;
                            }
                            else if(result.substring(0,1).equalsIgnoreCase("3")) //read shortcut
                                case
                            {
                                    out.println("Reading File...");
//                                  out.println("END");
                                    String fileName = result.substring(2, result.length());
                                    if (tokenMap.containsKey(fileName)){
                                            if (tokenOwner.get(fileName)==processID) {
                                                    out.println(IOFunctions.Read(tokenMap,
                                                        fileName));
                                                    out.println("SUCCESS: Read file \"" +
                                                        fileName + "\"\n");
//                                                  out.println("END");
                                            }
                                            else {
                                                    out.println("ERROR: You must have token to
                                                        read from file...\n");
                                                    out.println("The current token owner for file \"
                                                        " + fileName + "\" is \"" + tokenOwner.
                                                        get(fileName) + "\"...\n");
                                                    out.println("ACQUIRETOKEN");
//                                                  out.println("END");
                                            }
                                    }
                                    else {
                                            out.println("ERROR: File \"" + fileName + "\" does
                                                not exist in token map...\n");
//                                      out.println("END");
                                    }
                            }
                            else if(result.substring(0,1).equalsIgnoreCase("1")) //create shortcut
                                case
                            {
                                    out.println("Creating File ... ");
//                              out.println("END");
                                    String fileName = result.substring(2, result.length());
                                    if (fileName.contains(" ")){
                                            out.println("ERROR: Filename \"" + fileName + "\"
                                                cannot contain a space...\n");
//                                      out.println("END");
                                    }
                                    else if (tokenMap.containsKey(fileName)){
                                            out.println("ERROR: File \"" + fileName + "\" already
                                                 exists in token map...\n");
//                                      out.println("END");
                                    }
                                    else {
                                            IOFunctions.Create(tokenMap, fileName);
                                            tokenOwner.put(fileName, processID);
                                            out.println("SUCCESS: Created file \"" + fileName + "
                                                \" with owner \"" + tokenOwner.get(fileName) + "
                                                \"\n");
//                                      out.println("END");
                                    }

                            }
                            else if(result.substring(0,1).equalsIgnoreCase("2")) // delete shortcut
                                case
                            {
                                    out.println("Deleting File ... ");
//                              out.println("END");
                                    String fileName = result.substring(2, result.length());
                                    if (tokenMap.containsKey(fileName)){
                                            if (tokenOwner.get(fileName)==processID) {
                                                    IOFunctions.Delete(tokenMap, fileName);
                                                    out.println("SUCCESS: Deleted file \"" +
                                                        fileName + "\"\n");
//                                              out.println("END");
                                            }
                                            else {
                                                    out.println("ERROR: You must have token to
                                                        delete a file...\n");
                                                    out.println("The current token owner for file \"
                                                        " + fileName + "\" is \"" + tokenOwner.
                                            get(fileName) + "\"...\n");
                                        out.println("ACQUIRETOKEN");
//                                  out.println("END");
                                    }
                                }
                                else {
                                    out.println("ERROR: File \"" + fileName + "\" does
                                        not exist in token map...\n");
//                              out.println("END");
                                }
                            }
                            else if(result.substring(0,1).equalsIgnoreCase("4")) // append shortcut
                                case
                            {
                                out.println("Appending to File...");
//                          out.println("END");
                                String tmp = result.substring(2,result.length());
                                if(tmp.contains(" ")){
                                    int index = tmp.indexOf(' ');
                                    String fileName = tmp.substring(0,index);
                                    String line = tmp.substring(index+1,tmp.length());
                                    if (tokenMap.containsKey(fileName)){
                                        if(tokenOwner.get(fileName)==processID){
                                            IOFunctions.Append(tokenMap,
                                                fileName,line);
                                            out.println("SUCCESS: Appended line
                                                \"" + line + "\" to file \"" +
                                                fileName + "\"\n");
//                                      out.println("END");
                                        }
                                        else {
                                            out.println("ERROR: You must have
                                                token to append to a file...\n");
                                            out.println("The current token owner
                                                for file \"" + fileName + "\" is \""
                                                 + tokenOwner.get(fileName) + "
                                                \"...\n");
                                            out.println("ACQUIRETOKEN");
//                                      out.println("END");
                                        }
                                    } else {
                                        out.println("ERROR: File \"" + fileName + "
                                            \" does not exist in token map...\n");
//                                  out.println("END");
                                    }
                                } else {
                                    out.println("ERROR: Must include a line to append...\n
                                        ");
//                              out.println("END");
```

```
                    }
                }
                else if(result.substring(0,4).equalsIgnoreCase("read")) // read case
                {
                    out.println("Reading File...");
//                  out.println("END");
                    String fileName = result.substring(5, result.length());
                    if (tokenMap.containsKey(fileName)){
                        if (tokenOwner.get(fileName)==processID) {
                            out.println(IOFunctions.Read(tokenMap,
                                fileName));
                            out.println("SUCCESS: Read file \"" +
                                fileName + "\"\n");
//                          out.println("END");
                        }
                        else {
                            out.println("ERROR: You must have token to
                                read from file...\n");
                            out.println("The current token owner for file \"
                                " + fileName + "\" is \"" + tokenOwner.
                                get(fileName) + "\"...\n");
                            out.println("ACQUIRETOKEN");
//                          out.println("END");
                        }
                    }
                    else {
                        out.println("ERROR: File \"" + fileName + "\" does
                            not exist in token map...\n");
//                      out.println("END");
                    }
                }
                else if(result.substring(0,4).equalsIgnoreCase("list")) // list case
                {
                    for (String token : tokenMap.keySet()){
                        out.println("Token Name: \"" + token + "\"\nToken
                            Owner: \"" + tokenOwner.get(token) + "\"\
                            nContents: \n" + tokenMap.get(token) + "\n");
//                      out.println("END");
                    }
                }
                else if(result.substring(0,4).equalsIgnoreCase("exit")) // exit case
                {
                    out.println("Exiting...");
//                  out.println("END");
                    break;
                }
                else if(result.substring(0,6).equalsIgnoreCase("create")) // create case
                {
                    out.println("Creating File...");
```

```
//                                      out.println("END");
                                        String fileName = result.substring(7, result.length());
                                        if (fileName.contains(" ")){
                                                out.println("ERROR: Filename \"" + fileName + "\"
                                                    cannot contain a space...\n");
//                                              out.println("END");
                                        }
                                        else if (tokenMap.containsKey(fileName)){
                                                out.println("ERROR: File \"" + fileName + "\" already
                                                     exists in token map...\n");
//                                              out.println("END");
                                        }
                                        else {
                                                IOFunctions.Create(tokenMap, fileName);
                                                tokenOwner.put(fileName, processID);
                                                out.println("SUCCESS: Created file \"" + fileName + "
                                                    \" with owner \"" + tokenOwner.get(fileName) + "
                                                    \"\n");
//                                              out.println("END");
                                        }

                                }
                                else if(result.substring(0,6).equalsIgnoreCase("delete")) // delete case
                                {
                                        out.println("Deleting File ... ");
//                                      out.println("END");
                                        String fileName = result.substring(7, result.length());
                                        if (tokenMap.containsKey(fileName)){
                                                if (tokenOwner.get(fileName)==processID) {
                                                        IOFunctions.Delete(tokenMap, fileName);
                                                        out.println("SUCCESS: Deleted file \"" +
                                                            fileName + "\"\n");
//                                                      out.println("END");
                                                }
                                                else {
                                                        out.println("ERROR: You must have token to
                                                            delete a file...\n");
                                                        out.println("The current token owner for file \"
                                                            " + fileName + "\" is \"" + tokenOwner.
                                                            get(fileName) + "\"...\n");
                                                        out.println("ACQUIRETOKEN");
//                                                      out.println("END");
                                                }
                                        }
                                        else {
                                                out.println("ERROR: File \"" + fileName + "\" does
                                                    not exist in token map...\n");
//                                              out.println("END");
                                        }
```

```
                }
                else if(result.substring(0,6).equalsIgnoreCase("append")) // append
                    case
                {
                    out.println("Appending to File...");
//                  out.println("END");
                    String tmp = result.substring(7,result.length());
                    if(tmp.contains(" ")){
                        int index = tmp.indexOf(' ');
                        String fileName = tmp.substring(0,index);
                        String line = tmp.substring(index+1,tmp.length());
                        if (tokenMap.containsKey(fileName)){
                            if(tokenOwner.get(fileName)==processID){
                                IOFunctions.Append(tokenMap,
                                    fileName,line);
                                out.println("SUCCESS: Appended line
                                    \"" + line + "\" to file \"" +
                                    fileName + "\"\n");
//                              out.println("END");
                            }
                            else {
                                out.println("ERROR: You must have
                                    token to append to a file...\n");
                                out.println("The current token owner
                                    for file \"" + fileName + "\" is \""
                                     + tokenOwner.get(fileName) + "
                                    \"...\n");
                                out.println("ACQUIRETOKEN");
//                              out.println("END");
                            }
                        } else {
                            out.println("ERROR: File \"" + fileName + "
                                \" does not exist in token map...\n");
//                          out.println("END");
                        }
                    } else {
                        out.println("ERROR: Must include a line to append...\n
                            ");
//                      out.println("END");
                    }
                }
                else {
                    out.println("ERROR: Unknown Command...\n");
//                  out.println("END");
                }
            }
            //console.close();
        }
        }
}
```

II.VI Main

While the program can be run by using Client and Server classes, the main class can be used to run file operations locally. This class provided a sandbox to check the working status of many of the functionalities of this program.

```
/**
 *
 */
package distributed;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.Scanner;

import distributed.IOFunctions;
import distributed.MultiThread;
import raymonds.Process;
import raymonds.Tree;

/**
 * @author Sabbir Rashid
 *
 */
public class Main {

        /**
         * @param args
         * @throws Exception
         */
        public static void main(String[] args) throws Exception {
                // processes with neighbors derived from file
                ArrayList<Process> processes = Tree.CreateTree("tree.txt");
                // Print Process State
                PrintProcessStates(processes);
                // Token names list
                LinkedHashMap<String,String> tokenMap = new LinkedHashMap<String,String>();
                // Token owners list
                LinkedHashMap<String, String> tokenOwner = new LinkedHashMap<String,String>();
                //Process test = new Process("testID");
                //runIO(test, tokenMap, tokenOwner);

                for (Process p : processes) {
                        MultiThread.runMultiThread("5556",p.getProcessID());
                        runIO(p, tokenMap, tokenOwner);
                        PrintProcessStates(processes);
                }
                System.out.println("Program Finished Running...");
        }

        public static void PrintProcessStates(ArrayList<Process> processes){
                for (Process p : processes) {
                        System.out.println("Process ID:" + p.getProcessID());
                        System.out.println("Using Resource?:" + p.usingResource);
                        System.out.println("Asked?:" + p.asked);
                        System.out.println("Holder?:" + p.holderEnum.toString());
                        System.out.println("Neighbors:");
                        for (Process n : p.getNeighbors()) {
                                System.out.println("\t" + n.getProcessID());
                        }
                        System.out.println("Request Queue:");
                        for (Process r : p.requestQueue) {
                                System.out.println("\t" + r.getProcessID());
                        }
                        System.out.println("");
                }
        }

        public static void runIO(Process p, LinkedHashMap<String,String> tokenMap, LinkedHashMap
            <String,String> tokenOwner) throws IOException {
                // TODO Auto-generated method stub

                while(true) {
                        Scanner console = new Scanner(System.in);
                        System.out.println("Select the following command that you want to execute:");
                        System.out.println("1: create <filename>: creates an empty file named <
                            filename>");
                        System.out.println("2: delete <filename>: deletes file named <filename>");
                        System.out.println("3: read <filename>: displays the contents of <filename>");
                        System.out.println("4: append <filename> <line>: appends a <line> to <
                            filename>");
                        System.out.println("5: list : lists token owner and contents");
                        System.out.println("6: exit : exit");

                        if (console.hasNextLine()) {
                                String result = console.nextLine();
                                //Note: Calling create, delete, read, list , and append go here:
                                // Ordered by string length
                                if(result.equals("5")) // list shortcut
                                {
                                        for (String token : tokenMap.keySet()){
                                                System.out.println("Token Name: \"" + token + "\"\
                                                    nToken Owner: \"" + tokenOwner.get(token) + "
                                                    \"\nContents: \n" + tokenMap.get(token) + "\n");
            }
}
else if(result.equals("6")) // exit shortcut
{
        break;
}
else if(result.substring(0,1).equalsIgnoreCase("3")) //read shortcut
    case
{
        System.out.println("Reading File...");
        String fileName = result.substring(2, result.length());
        if (tokenMap.containsKey(fileName)){
                if (tokenOwner.get(fileName)==p.getProcessID()) {
                        System.out.println(IOFunctions.Read(tokenMap,
                            fileName));
                        System.out.println("SUCCESS: Read file \"" +
                            fileName + "\"\n");
                }
                else {
                        System.out.println("ERROR: You must have
                            token to read from file...\n");
                        System.out.println("The current token owner for
                            file \"" + fileName + "\" is \"" +
                            tokenOwner.get(fileName) + "\"...\n");
                }
        }
        else {
                System.out.println("ERROR: File \"" + fileName + "\"
                    does not exist in token map...\n");
        }
}
else if(result.substring(0,1).equalsIgnoreCase("1")) //create shortcut
    case
{
        System.out.println("Creating File ... ");
        String fileName = result.substring(2, result.length());
        if (fileName.contains(" ")){
                System.out.println("ERROR: Filename \"" + fileName
                    + "\" cannot contain a space...\n");
        }
        else if (tokenMap.containsKey(fileName)){
                System.out.println("ERROR: File \"" + fileName + "\"
                    already exists in token map...\n");
        }
        else {
                IOFunctions.Create(tokenMap, fileName);
                tokenOwner.put(fileName, p.getProcessID());
                System.out.println("SUCCESS: Created file \"" +
                    fileName + "\" with owner \"" + tokenOwner.get(
                fileName) + "\"\n");
        }

    }
    else if(result.substring(0,1).equalsIgnoreCase("2")) // delete shortcut
        case
    {
        System.out.println("Deleting File ... ");
        String fileName = result.substring(2, result.length());
        if (tokenMap.containsKey(fileName)){
            if (tokenOwner.get(fileName)==p.getProcessID()) {
                IOFunctions.Delete(tokenMap, fileName);
                System.out.println("SUCCESS: Deleted file \""
                    + fileName + "\"\n");
            }
            else {
                System.out.println("ERROR: You must have
                    token to delete a file...\n");
                System.out.println("The current token owner for
                    file \"" + fileName + "\" is \"" +
                    tokenOwner.get(fileName) + "\"...\n");
            }
        }
        else {
            System.out.println("ERROR: File \"" + fileName + "\"
                does not exist in token map...\n");
        }
    }
    else if(result.substring(0,1).equalsIgnoreCase("4")) // append shortcut
        case
    {
        System.out.println("Appending to File...");
        String tmp = result.substring(2, result.length());
        if(tmp.contains(" ")){
            int index = tmp.indexOf(' ');
            String fileName = tmp.substring(0,index);
            String line = tmp.substring(index+1,tmp.length());
            if (tokenMap.containsKey(fileName)){
                if(tokenOwner.get(fileName)==p.getProcessID()
                    ){
                    IOFunctions.Append(tokenMap,
                        fileName,line);
                    System.out.println("SUCCESS:
                        Appended line \"" + line + "\" to
                        file \"" + fileName + "\"\n");
                }
                else {
                    System.out.println("ERROR: You must
                        have token to append to a file...\n");
```

```
                                        System.out.println("The current token
                                            owner for file \"" + fileName + "
                                            \" is \"" + tokenOwner.get(
                                            fileName) + "\"...\n");
                                    }
                                } else {
                                    System.out.println("ERROR: File \"" +
                                        fileName + "\" does not exist in token map
                                        ...\n");
                                }
                            } else {
                                System.out.println("ERROR: Must include a line to
                                    append...\n");
                            }
                        }
                        else if(result.substring(0,4).equalsIgnoreCase("read")) // read case
                        {
                            System.out.println("Reading File...");
                            String fileName = result.substring(5, result.length());
                            if (tokenMap.containsKey(fileName)){
                                if (tokenOwner.get(fileName)==p.getProcessID()) {
                                    System.out.println(IOFunctions.Read(tokenMap,
                                        fileName));
                                    System.out.println("SUCCESS: Read file \"" +
                                        fileName + "\"\n");
                                }
                                else {
                                    System.out.println("ERROR: You must have
                                        token to read from file...\n");
                                    System.out.println("The current token owner for
                                        file \"" + fileName + "\" is \"" +
                                        tokenOwner.get(fileName) + "\"...\n");
                                }
                            }
                            else {
                                System.out.println("ERROR: File \"" + fileName + "\"
                                    does not exist in token map...\n");
                            }
                        }
                        else if(result.substring(0,4).equalsIgnoreCase("list")) // list case
                        {
                            for (String token : tokenMap.keySet()){
                                System.out.println("Token Name: \"" + token + "\"\
                                    nToken Owner: \"" + tokenOwner.get(token) + "
                                    \"\nContents: \n" + tokenMap.get(token) + "\n");
                            }
                        }
                        else if(result.substring(0,4).equalsIgnoreCase("exit")) // exit case
                        {
```

```
        break;
}
else if(result.substring(0,6).equalsIgnoreCase("create")) // create case
{
        System.out.println("Creating File ... ");
        String fileName = result.substring(7,result.length());
        if (fileName.contains(" ")){
                System.out.println("ERROR: Filename \"" + fileName
                    + "\" cannot contain a space...\n");
        }
        else if (tokenMap.containsKey(fileName)){
                System.out.println("ERROR: File \"" + fileName + "\"
                    already exists in token map...\n");
        }
        else {
                IOFunctions.Create(tokenMap, fileName);
                tokenOwner.put(fileName, p.getProcessID());
                System.out.println("SUCCESS: Created file \"" +
                    fileName + "\" with owner \"" + tokenOwner.get(
                    fileName) + "\"\n");
        }

}
else if(result.substring(0,6).equalsIgnoreCase("delete")) // delete case
{
        System.out.println("Deleting File ... ");
        String fileName = result.substring(7,result.length());
        if (tokenMap.containsKey(fileName)){
                if (tokenOwner.get(fileName)==p.getProcessID()) {
                        IOFunctions.Delete(tokenMap, fileName);
                        System.out.println("SUCCESS: Deleted file \""
                            + fileName + "\"\n");
                }
                else {
                        System.out.println("ERROR: You must have
                            token to delete a file...\n");
                        System.out.println("The current token owner for
                            file \"" + fileName + "\" is \"" +
                            tokenOwner.get(fileName) + "\"...\n");
                }
        }
        else {
                System.out.println("ERROR: File \"" + fileName + "\"
                    does not exist in token map...\n");
        }
}
else if(result.substring(0,6).equalsIgnoreCase("append")) // append
    case
{
```

```
                                        System.out.println("Appending to File...");
                                        String tmp = result.substring(7, result .length());
                                        if(tmp.contains(" ")){
                                                int index = tmp.indexOf(' ');
                                                String fileName = tmp.substring(0,index);
                                                String line = tmp.substring(index+1,tmp.length());
                                                if (tokenMap.containsKey(fileName)){
                                                        if(tokenOwner.get(fileName)==p.getProcessID()
                                                            ){
                                                                IOFunctions.Append(tokenMap,
                                                                    fileName,line);
                                                                System.out.println("SUCCESS:
                                                                    Appended line \"" + line + "\" to
                                                                    file \"" + fileName + "\"\n");
                                                        }
                                                        else {
                                                                System.out.println("ERROR: You must
                                                                    have token to append to a file...\n");
                                                                System.out.println("The current token
                                                                    owner for file \"" + fileName + "
                                                                    \" is \"" + tokenOwner.get(
                                                                    fileName) + "\"...\n");
                                                        }
                                                } else {
                                                        System.out.println("ERROR: File \"" +
                                                            fileName + "\" does not exist in token map
                                                            ...\n");
                                                }
                                        } else {
                                                System.out.println("ERROR: Must include a line to
                                                    append...\n");
                                        }
                                }
                                else
                                        System.out.println("ERROR: Unknown Command...\n");
                        }
                        //console.close();
                }

        }




}
```

III CONCLUSIONS

In conclusion, initially we were able to implement a Centralized File System and write the functionality for Raymond's algorithm. This approach proved to not adhere to the project guidelines, so a new approach was taken, where the system was designed in a distributed sense. While much of the Raymond algorithm functionality has been implemented, admittedly, there are still some debugs to be addressed. Nevertheless, this project provided a great learning opportunity into the implementation of Clients, Servers, Multiple Threads, and Distributed Systems. A great takeaway is that a project like this is indeed very time consuming, and extensive planning and debugging is required to achieve optimal results.