Contents

Raymonds Algorithm - A Centralized Approach

Sabbir Rashid & Rob Berman
Rensselaer Polytechnic Institute

I INTRODUCTION

In this report we discuss our implementation of Raymond's Algorithm, as well as our design for file operation, and our set up of Client and Server Access.

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.

Distributed Systems Project 1

To run the code complete the following steps.
Initial Set up
When reviewing your EC2 instance,
add a security rule to allow all port access.
Make sure you can reach the distributed networks
ping \<server_address\>
To install git
sudo yum install git
To update java
sudo yum install java-1.8.0
For compiling java
sudo yum install java-1.8.0-openjdk-devel
To remove older java
sudo yum remove java-1.7.0-openjdk
If you wish to keep the old java version,
use the following to determine the default versions to use
sudo /usr/sbin/alternatives --config java
And for the default compiler
sudo /usr/sbin/alternatives --config javac
To pull the git repository
git clone https://github.com/rashidsabbir/DistributedSysProject1.git

JAR Files
In the DistributedSysProject1 directory,
to run a Single Threaded Server instance,
(where other processes are blocked)
java -jar Server.jar \<port_number\>
To run a Multi-threaded Server instance
where multiple clients can access the system at a time
java -jar MultiThread.jar \<port_number\>
To run a Client instance
java -jar Client.jar \<host_address\> \<port_number\>

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

```
package raymonds;

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

public class Tree {
        private static ArrayList<Process> processes;
        public Tree() throws IOException
        {
                processes = new ArrayList<Process>();
                FileReader fr = new FileReader("tree.txt");
                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();
                }
        }
}
```

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

II.II.iii requestResource(Process p)

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

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

II.II.iv releaseResource(Process p)

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

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

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

II.II.vi receivedToken(Process p)

Assigns the token to the given function when it receives it

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

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.

```
package sockets;

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


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

        if (args.length != 2) {
            System.err.println(
                "Usage: java Client <host name> <port number>");
            System.exit(1);
        }

        String hostName = args[0];
        int portNumber = Integer.parseInt(args[1]);
        System.out.println("CLIENT: About to try to create Client Socket");
        try (
            Socket clientSocket = new Socket(hostName, portNumber);
                PrintWriter out =
                new PrintWriter(clientSocket.getOutputStream(), true);
                BufferedReader in =
                new BufferedReader(
                    new InputStreamReader(clientSocket.getInputStream()));
            BufferedReader stdIn =
                new BufferedReader(
                    new InputStreamReader(System.in));
                ){
                String userInput;
                System.out.println("CLIENT: About to wait for user input.");
                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: exit: exits the program");

                while ((userInput = stdIn.readLine()) != null) {
                        out.println(userInput);
                        System.out.println(in.readLine());
                        if (userInput.contains("read")){
                                String ans = "";
                                while(in.ready())
                    {
```

```
                                    ans=in.readLine();
                        System.out.println("CLIENT: In inner while loop.");
                        System.out.println(ans);
                    }
                    System.out.println("CLIENT: Exited inner while loop.");
                        }
                }
            System.out.println("CLIENT: Exited 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.

```
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 sever implementation allows for multiple clients to access the server.

```
package sockets;

import java.io.BufferedReader;
import java.io.File;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

import main.Main;

public class MultiThread {
        public static void main(String[] args) throws Exception {
                if (args.length != 1) {
                        System.err.println("Usage: java Server <port number>");
                        System.exit(1);
                }

                int portNumber = Integer.parseInt(args[0]);

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

                int id = 0;
                while (true) {
```

```
                    Socket clientSocket = m_ServerSocket.accept();
                    ClientServiceThread cliThread = new ClientServiceThread(clientSocket, id++);
                    cliThread.start();
                }
        }
}

class ClientServiceThread extends Thread {
        Socket clientSocket;
        int clientID = -1;
        boolean running = true;

        ClientServiceThread(Socket s, int i) {
                clientSocket = s;
                clientID = i;
        }

        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.");
                        //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();
                                running = false;
                                System.out.print("Stopping client thread for client : " +
                                    clientID);
                                Main.ExitConnection();
                        }
                        else
                                out.println("Error: Invalid Command");

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

II.VI Main

II.VI.i Create Tree Structure

Creates the tree structure from the given 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 as well.

```
package main;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;
import raymonds.Process;
```

```
public class Main {

        public static void main(String[] args) throws IOException {
                FileReader fr = new FileReader("tree.txt");
                String input = br.readLine();
                boolean first = true;
                ArrayList<Process> processes = new ArrayList<Process>();
                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();
        }
        for(int i=0;i<processes.size();i++)
        {
                System.out.println(processes.get(i).getProcessID());
                System.out.print("Neighbors: ");
                for(int j=0;j<processes.get(i).getNeighbors().size();j++)
                        System.out.print(processes.get(i).getNeighbors().get(j).getProcessID()+
                            " ");
                System.out.println();
        }
}
```

II.VI.ii Read, Create, Append & Delete

```
        public static File CreateFile( String fileName) throws IOException {

                File file = new File(fileName);
                file.createNewFile();
                return file;
        }

        public static void AppendFile( String fileName, String line) throws IOException {

                FileWriter writer = new FileWriter(fileName, true);
                writer.append(line + "\n");
                writer.flush();
                writer.close();

        }

        public static String ReadFile( String fileName) throws IOException {
                BufferedReader reader = new BufferedReader(new FileReader(fileName));
                String input = reader.readLine();
                String result = input;
                while(input!=null)
                {
                        input=reader.readLine();
                        if(input!=null)
                                result = result + "\n"+ input;
                }
                reader.close();
                return result;
        }

        public static void DeleteFile( String fileName) throws IOException {
                Runtime.getRuntime().exec(new String[]{"bash","-c","rm " + fileName});
        }

}
```

III CONCLUSIONS

In conclusion, while we were able to implement a Centralized File System and write the functionality for Raymond's algorithm, we did not complete the distributed approach.