Liya Norng

Project 2

2/21/16

For my I would pick Round Robin, because its generate a time sharing environment. User would love to have faster response time rather than wait for one job to be at a time. Its give fairness to all the process waiting in line since every process gets equal share of CPU time. Also the newly created process will be added to the end of the ready queue for executing. For example we it is shortest job first, and user came in last, that mean the use will wait for ever to get the CPU time. So that would be bad. Then for the STRF condition, it would also bad. For example if user want do something that requite really big burst time. Then in this condition, the user stuff will never get CPU time. But even with aging technique, it would take forever to bring up the priority to where it can get CPU time. So Round Robin would suite a way better out of the three-schedule method. Base on the output from the runs, even though Round Robin have higher waiting time and turn around time. It gave everyone to have fair CPU time. So no process would left behind.

Script started on Sun 21 Feb 2016 08:26:29 PM PST

[uh8425@algebra ~]$ javac GanttChart.java

[uh8425@algebra ~]$ java GanttChart

Please give me the file name.

t.txt

SJF Schedule

A 1 0

B 4 0

C 2 2

D 1 2

E 12 3

F 6 4

G 3 5

H 2 5

Submission: Process: Action:

0 A Process terminated

1 B Process terminated

5 D Process terminated

6 C Process terminated

8 H Process terminated

10 G Process terminated

13 F Process terminated

19 E Process terminated

31 Complete

Process ID: Turnaround Time: Waiting Time:

A 1 0

B 5 1

C 6 4

D 4 3

E 28 16

F 15 9

G 8 5

H 5 3

avg. 9.0 5.125

SRTF Schedule

A 1 0

B 4 0

C 2 2

D 1 2

E 12 3

F 6 4

G 3 5

H 2 5

Submission: Process: Action:

0 A Process terminated

1 B Process preempted

2 D Process terminated

3 C Process terminated

5 H Process terminated

7 B Process terminated

10 G Process terminated

13 F Process terminated

19 E Process terminated

31 Complete

Process ID: Turnaround Time: Waiting Time:

A 1 0

B 10 6

C 3 1

D 1 0

E 28 16

F 15 9

G 8 5

H 2 0

avg. 8.5 4.625

RR Schedule

A 1 0

B 4 0

C 2 2

D 1 2

E 12 3

F 6 4

G 3 5

H 2 5

Submission: Process: Action:

0 A Process terminated

1 B Process terminated

5 C Process terminated

7 D Process terminated

8 E Quatumn expired

12 F Quatumn expired

16 G Process terminated

19 H Process terminated

21 E Quatumn expired

25 F Process terminated

27 E Process terminated

31 Complete

Process ID: Turnaround Time: Waiting Time:

A 1 0

B 5 1

C 5 3

D 6 5

E 28 16

F 23 17

G 14 11

H 16 14

avg. 12.25 8.375

[uh8425@algebra ~]$ exit

exit

Script done on Sun 21 Feb 2016 08:26:53 PM PST

import java.util.\*;

import java.io.\*;

/\*\*

\* Liya Norng

\* 2/19/16

\* Operating System

\*/

public class GanttChart {

/\*\*

\* I need to parse the string right when i read the file so it easier for me to

\* the piece i want when i need. So i only need to parse the string once.

\*/

static LinkedList jobq = new LinkedList(); // this is for looping through and pop the queue out of the link

static LinkedList arriveTime = new LinkedList(); // this is linklist of arrive time

static LinkedList burstTime = new LinkedList(); // this is linklist for burst time

static LinkedList process = new LinkedList(); // this is linklist for keeping the process ID

static LinkedList second = new LinkedList(); // second linklist for Round Robin

static String[] s; // this is a temp. storage to keep the string when reading from file

static LinkedList list = new LinkedList(); // this is a linklist of no pasing the string

static Map<String,String> burstime = new HashMap<String, String>(); // this is my map for busrt time

static Map<String,String> arrive = new HashMap<String, String>(); // this is my map for arrive time

static Map<String,String> turnaround = new HashMap<String, String>(); // this is my map for turnaround time

static Map<String,String> waiting = new HashMap<String, String>(); // this is my map for waiting time

static Map<String,String> time = new HashMap<String, String>(); // this is the my map for storing burst time left

static ArrayList<String> nums = new ArrayList<String>(); // this is the string key to access the map which contains process ID

static ArrayList<String> ran = new ArrayList<String>(); // this is for outputing when the process have submit to the ready q

static Map<String,String> check = new HashMap<String, String>(); // this is for chekcing if a process have duplicate its time

public static void main(String[] args) {

Scanner input = new Scanner(System.in); // creating an instance of class for gettting the input from user

String fileName = null;

System.out.println("Please give me the file name."); // promping the user for file name.

fileName = input.nextLine(); // getting input from user

/\*\*

\* calling to the init to read from the file, and parse the string and put them in correct spot

\*/

GanttChart a = new GanttChart();

a.init(fileName);

jobq.addAll(list);

/\*\*

\* loop through and put all the linklist to the map

\*/

for (int i = 0; i < burstTime.size(); i++)

{

String s = (String)process.get(i);

int d = (int)burstTime.get(i);

time.put(s, String.valueOf(d));

arrive.put(s, String.valueOf(arriveTime.get(i)));

}

burstime.putAll(time);

/\*\*

\* looping through the jobq and pop the queue to the output window

\* displaying what is in the file

\*/

System.out.println("");

System.out.println("SJF Schedule");

while (true)

{

// if jobq is emtpy it will break the loop since im removing the jobq

if (jobq.isEmpty())

{

break;

}

System.out.println(jobq.pop());

}

jobq.addAll(list); // resetting the jobq for the next schedule

System.out.println("");

System.out.println("Submission: Process: Action:");

waiting.clear();

turnaround.clear();

int u = 0;

int spot = 0;

int j = 0;

int low = 0;

int i = 0;

int running = 0;

// looping through the event using time. so as time pass by i will store the process in the

// nums array to key the ID to access to the map. So without the key it mean that the process

// have not arrive during that time.

// this is for SJF Schedule

while (true)

{

// this is for checking for arrival of the process

for (String key: arrive.keySet())

{

if (j == Integer.parseInt(arrive.get(key)))

{

nums.add(key);

}

}

// since this is SJF, process won't preempted, i did a check for burst time = 0, then i pick another process in the

// list that has shortest job burst time

if (low == 0)

{

// adding the time when the process is in to have CPU time

ran.add(String.valueOf(j));

spot = 0; // using for accessing the nums to get the cerain process ID

u = 0; // my loop counter

low = Integer.parseInt(time.get(nums.get(spot))); // getting the next lowest slot in the array to compare

// this loop is for checking which process have the lowest time in the ready state

for (u = 0; u < nums.size(); u++)

{

// if someone else is lower than low then set low to that value, and get the location of the process in the array nums

if (Integer.parseInt(time.get(nums.get(u))) < low)

{

low = Integer.parseInt(time.get(nums.get(u)));

spot = u;

}

}

}

// so this is like the big action here. it take the value got earleir and mine - 1 since time hav pass by one.

low = low - 1;

time.remove(nums.get(spot)); // for this i remove the time in the slot and then

time.put(nums.get(spot), String.valueOf(low)); // add the to the same slot

// so if the low value, which is the burst time is 0, then it mean that the process have terminated

if (low == 0)

{

// this is for getting the arrive time so i can use to get the turnaround time.

int sub = Integer.parseInt(arrive.get(nums.get(spot)));

sub = j + 1 - sub;

turnaround.put(nums.get(spot), String.valueOf(sub)); // storing turnaround time to the map

// getting the burst time so i can substract the waiting time

int wait = sub - Integer.parseInt(burstime.get(nums.get(spot)));

waiting.put(nums.get(spot), String.valueOf(wait)); // storing the waiting time to the map

System.out.print(ran.get(running++));

System.out.print(" ");

System.out.print(nums.get(spot));

System.out.print(" ");

System.out.println("Process terminated");

ran.add(nums.get(spot));

nums.remove(spot);

running++;

// if it is empty it will display complete along with some other one too

if (nums.isEmpty())

{

System.out.print(j + 1);

System.out.print(" ");

System.out.println("Complete");

}

}

// it will break the loop if nums is empty since im removing them as the process is done

if (nums.isEmpty())

{

break;

}

j++; // loop counter

}

double waitingTime = 0; // use to store the total of waiting time

double turnAroundTime = 0; // use to store the total of turnaround time

i = 0;

// this will loop through and display the time for waiting and turnaround. as it loop it will add the time and store it

// in waiting time for waiting process, and turnaournd time for turn around.

System.out.println("");

System.out.println("Process ID: Turnaround Time: Waiting Time:");

for (String key: arrive.keySet())

{

System.out.print(key);

System.out.print(" ");

i = Integer.parseInt(turnaround.get(key));

System.out.print(i);

turnAroundTime = turnAroundTime + i;

i= 0;

i = Integer.parseInt(waiting.get(key));

System.out.print(" ");

System.out.println(i);

waitingTime = waitingTime + i;

i = 0;

}

// sending output to the window for the average time for turnaround and waiting time

System.out.print("avg.");

System.out.print(" ");

System.out.print(turnAroundTime/arrive.size());

System.out.print(" ");

System.out.println(waitingTime/arrive.size());

// resetting the array, and map for the next scheduling

arrive.clear();

time.clear();

nums.clear();

ran.clear();

turnaround.clear();

waiting.clear();

i = 0;

for (i = 0; i < burstTime.size(); i++)

{

String s = (String)process.get(i);

int d = (int)burstTime.get(i);

time.put(s, String.valueOf(d));

arrive.put(s, String.valueOf(arriveTime.get(i)));

}

System.out.println("");

System.out.println("SRTF Schedule");

// same thing again it will loop through the jobq and output the whole line of string that read from the file

while (true)

{

if (jobq.isEmpty())

{

break;

}

System.out.println(jobq.pop());

}

jobq.addAll(list); // resetting the jobq for the next schedule

System.out.println("");

System.out.println("Submission: Process: Action:");

// resetting its value to 0 getting ready for the next loop

j = 0;

u = 0;

spot = 0;

low = 0;

running = 0;

int done = 0;

int add = 0;

int complete = 0;

// this is for SRTF schedule

while (true)

{ // gettting the process that arrive during j as time pass by. And j is my time counter

for (String key: arrive.keySet())

{ // if it matches with j, it will add the process to the nums array

if (j == Integer.parseInt(arrive.get(key)))

{

nums.add(key);

check.put(key, "0");

}

}

// this will only run if the process is terminate. i have to use if else statem for this to get the time for a process

// is terminate.

if (low == 0)

{

u = 0; // loop counter

spot = 0; // using for checking where in the nums is the next process

low = Integer.parseInt(time.get(nums.get(0))); // getting the lowest slot in the nums array.

// looping through to get the lowest burst time of the process that arrive duing j time

for (u = 0; u < nums.size(); u++)

{ // if it lower than low, it will set it to low and set the location of the nums array to spot

if ( Integer.parseInt(time.get(nums.get(u))) < low)

{

low = Integer.parseInt(time.get(nums.get(u)));

spot = u;

}

}

ran.add(String.valueOf(j)); // adding the time it terminate

}

else

{

u = 0; // loop counter

spot = 0; // using for checking where in the nums is the next process

low = Integer.parseInt(time.get(nums.get(0))); // getting the lowest slot in the nums array.

// looping through to get the lowest burst time of the process that arrive duing j time

for (u = 0; u < nums.size(); u++)

{ // if it lower than low, it will set it to low and set the location of the nums array to spot

if ( Integer.parseInt(time.get(nums.get(u))) < low)

{

low = Integer.parseInt(time.get(nums.get(u)));

spot = u;

}

}

}

// this is where the big action happen. it take the low which is the burst time - 1.

low = low - 1;

time.remove(nums.get(spot)); // removing the map and then add it back to the map

time.put(nums.get(spot), String.valueOf(low));

// this will run if the process burst time is 0

if (low == 0) // this if else statement is use for determine for terminate state and preempted state. Also complete state

{

complete = 1;

done++; // counting if it near ot the last process then it will not print out preempted

int sub = Integer.parseInt(arrive.get(nums.get(spot))); // getting the arrive time to get the turnaround time

sub = j + 1 - sub;

turnaround.put(nums.get(spot), String.valueOf(sub)); // storing the turnaround time

int wait = sub - Integer.parseInt(burstime.get(nums.get(spot))); // getting the burst time and did some substraction

waiting.put(nums.get(spot), String.valueOf(wait)); // storing the waiting time

add = add + Integer.parseInt(ran.get(running++)); // this is use for the time when the process have submit to the ready state

System.out.print(add);

add = 0;

System.out.print(" ");

System.out.print(nums.get(spot));

System.out.print(" ");

System.out.println("Process terminated");

nums.remove(spot);

if (nums.isEmpty())

{

System.out.print(j + 1);

System.out.print(" ");

System.out.println("Complete");

}

}

else // since this is SRTF it will preempted any process that has lower than the one that is running.

{ // this is use to check for duplicaiton of the process saying preempted

if (check.containsKey(nums.get(spot)))

{

int k = 0;

k = 1;

check.remove(nums.get(spot));

check.put(nums.get(spot), String.valueOf(k));

}

int locat = 0; // storing the last spot to compare if it the same

// i needs so many if statement since the slot of the nums array is change for certain process. some are terminate as time

// pass by. So i have three if statement checking for duplicate of the process to say preempted even tho it is still running

// in CPU.

if (check.containsKey(nums.get(spot)))

{

if (spot == locat) // reason y it check for == because it as time pass by it will have more process, and some are going

{ // be terminate. so the slot will not be in the same location.

if (done != arrive.size() - 1)

{

add = add + 1; // this is the counter for the submission of the process is sending back to the terminate if statement

System.out.print(j);

System.out.print(" ");

System.out.print(nums.get(spot));

System.out.print(" ");

System.out.println("Process preempted");

check.remove(nums.get(spot));

}

}

locat = spot; // setting the current spot to the locat to store for comparing when it come around again

}

}

j++; // loop counter

// will end if the nums is empty.

if (nums.isEmpty())

{

break;

}

}

// resetting the value to 0 for getting ready to add the waiting and turnaround time.

waitingTime = 0;

turnAroundTime = 0;

i = 0;

System.out.println("");

System.out.println("Process ID: Turnaround Time: Waiting Time:");

for (String key: arrive.keySet()) // looping through the turnaround time and waiting time to get the avg, and display

{ // diplay the waiting time and turnaround time to the window

System.out.print(key);

System.out.print(" ");

i = Integer.parseInt(turnaround.get(key));

System.out.print(i);

// calculation for the turnAround time

turnAroundTime = turnAroundTime + i;

i= 0;

// calculation for the waiting time

i = Integer.parseInt(waiting.get(key));

System.out.print(" ");

System.out.println(i);

waitingTime = waitingTime + i;

i = 0;

}

System.out.print("avg.");

System.out.print(" ");

System.out.print(turnAroundTime/arrive.size());

System.out.print(" ");

System.out.println(waitingTime/arrive.size());

// resetting for the next schedule

arrive.clear();

time.clear();

nums.clear();

check.clear();

ran.clear();

turnaround.clear();

waiting.clear();

// resetting its value to the map again. I don't really have to reset the value of the map

// but its for preventing from getting bugs since im move onto difference platform

i = 0;

for (i = 0; i < burstTime.size(); i++)

{

String s = (String)process.get(i);

int d = (int)burstTime.get(i);

time.put(s, String.valueOf(d));

arrive.put(s, String.valueOf(arriveTime.get(i)));

}

System.out.println("");

System.out.println("RR Schedule");

while (true)

{

if (jobq.isEmpty())

{

break;

}

System.out.println(jobq.pop());

}

jobq.addAll(list);

process.clear();

System.out.println("");

System.out.println("Submission: Process: Action:");

j = 0;

spot = 0;

low = 0;

int termin = 0;

running =0;

int total = 0;

i = 0;

// Round Robin Schedule

while (true)

{ // getting the process that arrive at time j

for (String key: arrive.keySet())

{

if (j == Integer.parseInt(arrive.get(key)))

{

nums.add(key);

}

}

// this is my if statement for checking if it reaches 4 quatumn

if (total == 4)

{ // if it true, then it will remove from nums, and add into the second array

second.add(nums.remove(spot));

ran.add(String.valueOf(j)); // this is use for storing the time of submission

total = 0; // resetting the quatumn time

}

// checking if nums is empty. if it is it will get the left over process from second arry and put it back to nums

// clear the second array to store again for expired quatumn that process never terminate

if (nums.isEmpty())

{

nums.addAll(second);

second.clear();

}

// if the burst time is 0, it will get next process of the list

// this if statement is use for getting the time when the process submission

if (low == 0)

{

spot = 0;

low = Integer.parseInt(time.get(nums.get(0))); // gettin the next process since it all in order

ran.add(String.valueOf(j)); // adding the time next process is running.

termin = 1;

}

else

{

spot = 0;

low = Integer.parseInt(time.get(nums.get(0))); // gettin the next process since it all in order

}

// doing some calculation for burst time.

low = low - 1;

time.remove(nums.get(spot)); // removeing from the map time. and add it back to the map time

time.put(nums.get(spot), String.valueOf(low));

total ++; // this is my quatumn counter

if (low == 0) // this is when the burst time is 0

{ // doing some calculation for the turnaround time and waiting time

int sub = Integer.parseInt(arrive.get(nums.get(spot)));

sub = j + 1 - sub;

turnaround.put(nums.get(spot), String.valueOf(sub));

int wait = sub - Integer.parseInt(burstime.get(nums.get(spot)));

waiting.put(nums.get(spot), String.valueOf(wait));

if (j == 0)

{

System.out.print(j); // this is print out the process at time 0. since there is always a process in the CPU

}

else

{

System.out.print(ran.get(++running)); // output the submission time to the window

}

System.out.print(" ");

System.out.print(nums.get(spot));

System.out.print(" ");

System.out.println("Process terminated");

nums.remove(spot);

// this is a check if nums, and the second array empty.

if (nums.isEmpty() && second.isEmpty())

{

System.out.print(j + 1);

System.out.print(" ");

System.out.println("Complete");

}

total = 0; // resetting the quatumn time

}

else

{

// this is for checking if the quatumn time is finish if it is it will output the submission time, process ID, and quatumn expired

if (total == 4)

{

System.out.print(ran.get(++running));

System.out.print(" ");

System.out.print(nums.get(spot));

System.out.print(" ");

System.out.println("Quatumn expired");

}

}

j++; // loop counter for using as time event

// this is use for breaking the loop if both of is empty

if (nums.isEmpty() && second.isEmpty())

{

break;

}

}

// resetting it value to 0

waitingTime = 0;

turnAroundTime = 0;

i = 0;

System.out.println("");

System.out.println("Process ID: Turnaround Time: Waiting Time:");

// looping through waiting time, and turnaournd time display the output

for (String key: arrive.keySet())

{

System.out.print(key);

System.out.print(" ");

i = Integer.parseInt(turnaround.get(key));

turnAroundTime = turnAroundTime + i; // some calculation of turnAroundTime

System.out.print(i);

i= 0;

i = Integer.parseInt(waiting.get(key));

System.out.print(" ");

System.out.println(i);

waitingTime = waitingTime + i; // some calculation of waiting time

i = 0;

}

// this is for the average waiting time, and turnaround time

System.out.print("avg.");

System.out.print(" ");

System.out.print(turnAroundTime/arrive.size());

System.out.print(" ");

System.out.println(waitingTime/arrive.size());

}

// this is for reading from the file

public void init(String fileName)

{ // so i use try and catch. so it will try to open and read the file its fail, it will display an error message

try

{

// this is to read the file, and from there it store to the string above for temperary.

FileReader file\_Input = new FileReader(fileName);

BufferedReader buffer = new BufferedReader(file\_Input); // buffer

String line = null;

// this is a while loop, and reading a line at a time, and store each line to appropriate area.

while((line = buffer.readLine()) != null)

{

// parsing the string and separte value to appropriate list

s = line.split(" "); // spliting the string by space

list.add(line);

arriveTime.add((Integer.parseInt(s[2])));

burstTime.add((Integer.parseInt(s[1])));

process.add(s[0]);

}

buffer.close();

}// this is use to catch the error

catch (IOException ex)

{

System.out.println("Sorry, file u.txt does not exist.");

}

}

}