

**WIA2004 OPERATING SYSTEM**

**LAB 2 PROJECT REPORT**

**CPU**

**GROUP MEMBERS**

| **No.** | **Member Name** | **Matric Number** |
| --- | --- | --- |
| **1.** | **MUHAMMAD IZZAT HAKEEM BIN ROSDI** | **17204856** |
| **2.** | **WAN MUHAMMAD AZIB BIN WAN ABDUL RAHIM** | **17207678** |
| **3.** | **MUHAMMAD AMIRUL AMIN BIN ARMAN** | **17202969** |
| **4.** | **JONE WAN GAN** | **S2031153** |
| **5.** | **LAWRENCE LEROY CHIENG TZE YAO** | **S2018935** |

### **QUESTION**

Write a program to simulate the following non-preemptive CPU 1 scheduling algorithms to find turnaround time and waiting time for SJF.

### **OBJECTIVE**

Write a program to simulate the following non-preemptive CPU scheduling algorithms to find turnaround time and waiting time for the above problem.

### **DESCRIPTION**

Assume all the processes arrive at the same time.

### **SJF CPU SCHEDULING ALGORITHM**

For the SJF scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times. Arrange all the jobs in order with respect to their burst times. There may be two jobs in queue with the same execution time, and then FCFS approach is to be performed. Each process will be executed according to the length of its burst time. Then calculate the waiting time and turnaround time of each of the processes accordingly.

### **SJF**

* + SJF is an algorithm in which the process having the smallest execution time is chosen for the next execution.
  + SJF Scheduling is associated with each job as a unit of time to complete.
  + This algorithm method is helpful for batch-type processing, where waiting for jobs to complete is not critical.
  + There are basically two types of SJF methods 1) Non-Preemptive SJF and 2) Preemptive SJF.
  + In non-preemptive scheduling, once the CPU cycle is allocated to a process, the process holds it till it reaches a waiting state or is terminated.
  + In Preemptive SJF Scheduling, jobs are put into the ready queue as they come.
  + Although a process with short burst time begins, the current process is removed or preempted from execution, and the job which is shorter is executed 1st.
  + SJF is frequently used for long term scheduling.
  + It reduces the average waiting time over FIFO (First in First Out) algorithm.
  + In SJF scheduling, Job completion time must be known earlier, but it is hard to predict.
  + SJF can’t be implemented for CPU scheduling for the short term. It is because there is no specific method to predict the length of the upcoming CPU burst.

### **CODE**

import java.util.Scanner;

public class sjf {

public static void main(String[] args) {

Scanner sc = new Scanner(System.*in*);

String input;

System.*out*.print("Enter the processes and their burst time(ex: p1,12,p2,23,p3,9,p4,30): ");

input = sc.next();

// deciding the number of processes based on ','

int size = 1;

for (int i = 0; i < input.length(); i++) {

if (input.charAt(i) == ',') {

size++;

}

}

size /= 2;

// convert input String to 2D array

String[] inputArr = new String[size \* 2];

String[][] input2Arr = new String[size][2];

int k = 0;

inputArr = input.split(",");

for (int i = 0; i < inputArr.length; i += 2) {

input2Arr[k++][0] = inputArr[i];

}

k = 0;

for (int i = 1; i < inputArr.length; i += 2) {

input2Arr[k++][1] = inputArr[i];

}

// sort for SJF

String temp1 = "";

String temp2 = "";

for (int pass = 0; pass < size - 1; pass++) {

for (int i = 1; i < size; i++) {

if (Integer.*parseInt*(input2Arr[i][1]) < Integer.*parseInt*(input2Arr[i - 1][1])) {

temp1 = input2Arr[i][0];

temp2 = input2Arr[i][1];

input2Arr[i][0] = input2Arr[i - 1][0];

input2Arr[i][1] = input2Arr[i - 1][1];

input2Arr[i - 1][0] = temp1;

input2Arr[i - 1][1] = temp2;

}

}

}

// for output

int[] waitingTime = new int[size];

int[] turnAround = new int[size];

waitingTime = *getWaiting*(input2Arr, size);

turnAround = *getTurnAround*(input2Arr, size);

System.*out*.println("Output:\nProcesses\tBurst\tWaiting\tTurn Around");

for (int i = 0; i < size; i++) {

System.*out*

.println(input2Arr[i][0] + "\t\t" + input2Arr[i][1] + "\t" + waitingTime[i] + "\t" + turnAround[i]);

}

System.*out*.println("Average Waiting Time = " + *getWaitingAvg*(waitingTime) + "\nAverage Turn Around Time: "

+ *getTurnAroundAvg*(turnAround));

}

// Turnaround Time = completion of a process – submission of a process

// Waiting Time = turnaround time – burst time

// Waiting Time method

public static int[] getWaiting(String[][] inputArray, int size) {

int[] waitingTime = new int[size];

int waitingValue = 0;

waitingTime[0] = 0;

for (int i = 1; i < size; i++) {

waitingValue += Integer.*parseInt*(inputArray[i - 1][1]);

waitingTime[i] = waitingValue;

}

return waitingTime;

}

// Average Waiting Time method

public static double getWaitingAvg(int[] waitingTime) {

double sum = 0, avg;

for (int i = 0; i < waitingTime.length; i++) {

sum += waitingTime[i];

}

avg = sum / waitingTime.length;

return avg;

}

// Turn Around time method

public static int[] getTurnAround(String[][] inputArray, int size) {

int[] turnAround = new int[size];

int turnAroundValue = 0;

for (int i = 0; i < size; i++) {

turnAroundValue += Integer.*parseInt*(inputArray[i][1]);

turnAround[i] += turnAroundValue;

}

return turnAround;

}

// AVerage Turn Around time method

public static double getTurnAroundAvg(int[] turnAround) {

double sum = 0, avg;

for (int i = 0; i < turnAround.length; i++) {

sum += turnAround[i];

}

avg = sum / turnAround.length;

return avg;

}

}

### **OUTPUT**

