# Answer 01:

## Code:

#!/bin/bash

echo "Input the string without space"

read str

for i in $(seq 0 ${#str}) ; do

revstr=${str:$i:1}$revstr

done

echo "The given string is " $str

if [ "$str" = "$revstr" ]; then

echo "It is a palindrome."

else

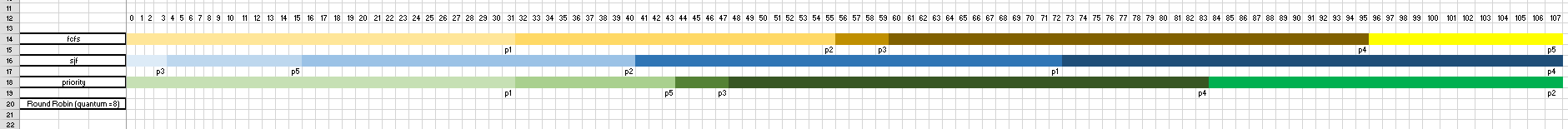
echo "It is not a palindrome."

fi

# Answer 02:

## Gantt Charts:





## Codes:

### FCFS

// Java program for implementation of FCFS scheduling

import java.text.ParseException;

class GFG {

// Function to find the waiting time for all processes

static void findWaitingTime(int processes[], int n,

int bt[], int wt[]) {

// waiting time for first process is 0

wt[0] = 0;

// calculating waiting time

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

wt[i] = bt[i - 1] + wt[i - 1];

}

}

// Function to calculate turn around time

static void findTurnAroundTime(int processes[], int n,

int bt[], int wt[], int tat[]) {

// calculating turnaround time by adding

// bt[i] + wt[i]

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

tat[i] = bt[i] + wt[i];

}

}

//Function to calculate average time

static void findavgTime(int processes[], int n, int bt[]) {

int wt[] = new int[n], tat[] = new int[n];

int total\_wt = 0, total\_tat = 0;

//Function to find waiting time of all processes

findWaitingTime(processes, n, bt, wt);

//Function to find turn around time for all processes

findTurnAroundTime(processes, n, bt, wt, tat);

//Display processes along with all details

System.out.printf("Processes Burst time Waiting"

+" time Turn around time\n");

// Calculate total waiting time and total turn around time

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

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

System.out.printf(" %d ", (i + 1));

System.out.printf(" %d ", bt[i]);

System.out.printf(" %d", wt[i]);

System.out.printf(" %d\n", tat[i]);

}

float s = (float)total\_wt /(float) n;

int t = total\_tat / n;

System.out.printf("Average waiting time = %f", s);

System.out.printf("\n");

System.out.printf("Average turn around time = %d ", t);

}

// Driver code

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

//process id's

int processes[] = {1, 2, 3};

int n = processes.length;

//Burst time of all processes

int burst\_time[] = {10, 5, 8};

findavgTime(processes, n, burst\_time);

}

}

### SJF

// Java program to implement Shortest Job first with Arrival Time

import java.util.\*;

class GFG {

static int[][] mat = new int[10][6];

static void arrangeArrival(int num, int[][] mat) {

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

for (int j = 0; j < num - i - 1; j++) {

if (mat[j][1] > mat[j + 1][1]) {

for (int k = 0; k < 5; k++) {

int temp = mat[j][k];

mat[j][k] = mat[j + 1][k];

mat[j + 1][k] = temp;

}

}

}

}

}

static void completionTime(int num, int[][] mat) {

int temp, val = -1;

mat[0][3] = mat[0][1] + mat[0][2];

mat[0][5] = mat[0][3] - mat[0][1];

mat[0][4] = mat[0][5] - mat[0][2];

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

temp = mat[i - 1][3];

int low = mat[i][2];

for (int j = i; j < num; j++) {

if (temp >= mat[j][1] && low >= mat[j][2]) {

low = mat[j][2];

val = j;

}

}

mat[val][3] = temp + mat[val][2];

mat[val][5] = mat[val][3] - mat[val][1];

mat[val][4] = mat[val][5] - mat[val][2];

for (int k = 0; k < 6; k++) {

int tem = mat[val][k];

mat[val][k] = mat[i][k];

mat[i][k] = tem;

}

}

}

// Driver Code

public static void main(String[] args) {

int num;

Scanner sc = new Scanner(System.in);

System.out.println("Enter number of Process: ");

num = sc.nextInt();

System.out.println("...Enter the process ID...");

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

System.out.println("...Process " + (i + 1) + "...");

System.out.println("Enter Process Id: ");

mat[i][0] = sc.nextInt();

System.out.println("Enter Arrival Time: ");

mat[i][1] = sc.nextInt();

System.out.println("Enter Burst Time: ");

mat[i][2] = sc.nextInt();

}

System.out.println("Before Arrange...");

System.out.println("Process ID\tArrival Time\tBurst Time");

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

System.out.printf("%d\t\t%d\t\t%d\n",

mat[i][0], mat[i][1], mat[i][2]);

}

arrangeArrival(num, mat);

completionTime(num, mat);

System.out.println("Final Result...");

System.out.println("Process ID\tArrival Time\tBurst" +

" Time\tWaiting Time\tTurnaround Time");

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

System.out.printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n",

mat[i][0], mat[i][1], mat[i][2], mat[i][4], mat[i][5]);

}

sc.close();

}

}

### Priority

import java.util.\*;

// Data Structure

class Process {

int at, bt, pri, pno;

Process(int pno, int at, int bt, int pri)

{

this.pno = pno;

this.pri = pri;

this.at = at;

this.bt = bt;

}

}

/// Gantt chart structure

class GChart {

// process number, start time, complete time, turn around time, waiting time

int pno, stime, ctime, wtime, ttime;

}

// user define comparative method (first arrival first serve,

// if arrival time same then heigh priority first)

class MyComparator implements Comparator {

public int compare(Object o1, Object o2)

{

Process p1 = (Process)o1;

Process p2 = (Process)o2;

if (p1.at < p2.at)

return (-1);

else if (p1.at == p2.at && p1.pri > p2.pri)

return (-1);

else

return (1);

}

}

// class to find Gantt chart

class FindGantChart {

void findGc(LinkedList queue)

{

// initial time = 0

int time = 0;

// priority Queue sort data according

// to arrival time or priority (ready queue)

TreeSet prique = new TreeSet(new MyComparator());

// link list for store processes data

LinkedList result = new LinkedList();

// process in ready queue from new state queue

while (queue.size() > 0)

prique.add((Process)queue.removeFirst());

Iterator it = prique.iterator();

// time set to according to first process

time = ((Process)prique.first()).at;

// scheduling process

while (it.hasNext()) {

// dispatcher dispatch the

// process ready to running state

Process obj = (Process)it.next();

GChart gc1 = new GChart();

gc1.pno = obj.pno;

gc1.stime = time;

time += obj.bt;

gc1.ctime = time;

gc1.ttime = gc1.ctime - obj.at;

gc1.wtime = gc1.ttime - obj.bt;

/// store the exxtreted process

result.add(gc1);

}

// create object of output class and call method

new ResultOutput(result);

}

}

### Round Robin

// Java program for implementation of RR scheduling

public class GFG

{

// Method to find the waiting time for all

// processes

static void findWaitingTime(int processes[], int n,

int bt[], int wt[], int quantum)

{

// Make a copy of burst times bt[] to store remaining

// burst times.

int rem\_bt[] = new int[n];

for (int i = 0 ; i < n ; i++)

rem\_bt[i] = bt[i];

int t = 0; // Current time

// Keep traversing processes in round robin manner

// until all of them are not done.

while(true)

{

boolean done = true;

// Traverse all processes one by one repeatedly

for (int i = 0 ; i < n; i++)

{

// If burst time of a process is greater than 0

// then only need to process further

if (rem\_bt[i] > 0)

{

done = false; // There is a pending process

if (rem\_bt[i] > quantum)

{

// Increase the value of t i.e. shows

// how much time a process has been processed

t += quantum;

// Decrease the burst\_time of current process

// by quantum

rem\_bt[i] -= quantum;

}

// If burst time is smaller than or equal to

// quantum. Last cycle for this process

else

{

// Increase the value of t i.e. shows

// how much time a process has been processed

t = t + rem\_bt[i];

// Waiting time is current time minus time

// used by this process

wt[i] = t - bt[i];

// As the process gets fully executed

// make its remaining burst time = 0

rem\_bt[i] = 0;

}

}

}

// If all processes are done

if (done == true)

break;

}

}

// Method to calculate turn around time

static void findTurnAroundTime(int processes[], int n,

int bt[], int wt[], int tat[])

{

// calculating turnaround time by adding

// bt[i] + wt[i]

for (int i = 0; i < n ; i++)

tat[i] = bt[i] + wt[i];

}

// Method to calculate average time

static void findavgTime(int processes[], int n, int bt[],

int quantum)

{

int wt[] = new int[n], tat[] = new int[n];

int total\_wt = 0, total\_tat = 0;

// Function to find waiting time of all processes

findWaitingTime(processes, n, bt, wt, quantum);

// Function to find turn around time for all processes

findTurnAroundTime(processes, n, bt, wt, tat);

// Display processes along with all details

System.out.println("Processes " + " Burst time " +

" Waiting time " + " Turn around time");

// Calculate total waiting time and total turn around time

for (int i=0; i<n; i++)

{

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

System.out.println(" " + (i+1) + "\t\t" + bt[i] +"\t " +

wt[i] +"\t\t " + tat[i]);

}

System.out.println("Average waiting time = " +

(float)total\_wt / (float)n);

System.out.println("Average turn around time = " +

(float)total\_tat / (float)n);

}

// Driver Method

public static void main(String[] args)

{

// process id's

int processes[] = { 1, 2, 3};

int n = processes.length;

// Burst time of all processes

int burst\_time[] = {10, 5, 8};

// Time quantum

int quantum = 2;

findavgTime(processes, n, burst\_time, quantum);

}

}