This code is written in Java and implements the Shortest Remaining Time First (SRTF) scheduling algorithm for processes. It takes the number of processes, their arrival time and burst time as input, and calculates their completion time, turn around time and waiting time. It also prints the average turn around time and waiting time for all processes

import java.util.\*;

public class SRTF {

public static void main (String args[])

{

// Create a Scanner object to read input from the user

Scanner sc=new Scanner(System.in);

System.out.println ("enter no of process:");

int n= sc.nextInt(); // Read the number of processes from the user

int pid[] = new int[n]; // Array to store process IDs

int at[] = new int[n]; // Array to store arrival times of processes

int bt[] = new int[n]; // Array to store burst times of processes

int ct[] = new int[n]; // Array to store completion times of processes

int ta[] = new int[n]; // Array to store turnaround times of processes

int wt[] = new int[n]; // Array to store waiting times of processes

int f[] = new int[n]; // Array to flag whether a process is completed or not

int k[]= new int[n]; // Array to store a copy of burst times of processes

int i, st=0, tot=0;

float avgwt=0, avgta=0;

// Loop to input process details from the user

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

{

pid[i]= i+1; // Assign a unique process ID

System.out.println ("enter process " +(i+1)+ " arrival time:");

at[i]= sc.nextInt(); // Read arrival time of the process

System.out.println("enter process " +(i+1)+ " burst time:");

bt[i]= sc.nextInt(); // Read burst time of the process

k[i]= bt[i]; // Copy burst time to 'k' array for later use

f[i]= 0; // Initialize the flag to indicate that the process is not completed yet

}

// Main scheduling loop using Shortest Remaining Time First (SRTF) algorithm

while(true){

int min=99,c=n;

if (tot==n)

break; // Exit the loop when all processes are completed

// Find the process with the shortest remaining burst time in the ready queue

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

{

if ((at[i]<=st) && (f[i]==0) && (bt[i]<min))

{

min=bt[i];

c=i; // Store the index of the process with the shortest remaining burst time

}

}

if (c==n)

st++; // If no process is available for execution, increment the system time

else

{

bt[c]--; // Decrement the remaining burst time of the selected process

st++; // Increment the system time

if (bt[c]==0)

{

ct[c]= st; // Store the completion time of the process

f[c]=1; // Set the flag to indicate that the process is completed

tot++; // Increment the count of completed processes

}

}

}

// Calculate turnaround time and waiting time for each process

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

{

ta[i] = ct[i] - at[i]; // Turnaround time = Completion time - Arrival time

wt[i] = ta[i] - k[i]; // Waiting time = Turnaround time - Original burst time

avgwt+= wt[i]; // Sum of waiting times for later calculation of average waiting time

avgta+= ta[i]; // Sum of turnaround times for later calculation of average turnaround time

}

// Print the result table showing process details and metrics

System.out.println("pid arrival burst complete turn waiting");

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

{

System.out.println(pid[i] +"\t"+ at[i]+"\t"+ k[i] +"\t"+ ct[i] +"\t"+ ta[i] +"\t"+ wt[i]);

}

// Print the average turnaround time and average waiting time

System.out.println("\naverage tat is "+ (float)(avgta/n));

System.out.println("average wt is "+ (float)(avgwt/n));

sc.close(); // Close the Scanner object to free up resources

}

}