**LOVELY PROFESSIONAL UNIVERSITY  
Academic Task-3 (Operating System)**

School of Computer Science and Engineering Faculty of Technology & Sciences

Name of the faculty member: Ashu

Course Code: CSE 316 Course Title: Operating System   
Max. Marks: 30 Date of Allotment: 29/02/2020

**Student Name: Vaibhav Dilip Sawandre**

**Student ID : 11805312**

**Email Address: vaibhavsawandre7@gmail.com**

**GitHub Link: https://github.com/klaus1002/OS\_CA\_Project.git**

**Problem** :

**Ques. 4.**  Consider a scheduling approach which is non pre-emptive similar to shortest job next in nature. The priority of each job is dependent on its estimated run time, and also the amount of time it has spent waiting. Jobs gain higher priority the longer they wait, which prevents indefinite postponement. The jobs that have spent a long time waiting compete against those estimated to have short run times. The priority can be computed as :

Priority = 1+ Waiting time / Estimated run time

Using the data given below compute the waiting time and turnaround time for each process and average waiting time and average turnaround time.

|  |  |  |
| --- | --- | --- |
| Process | Arrival time | Burst time |
| P1 | 0 | 20 |
| P2 | 5 | 36 |
| P3 | 13 | 19 |
| P4 | 17 | 42 |

**Description of concept :**

Shortest job first is a scheduling algorithm in which the process with the smallest execution time is selected for execution next. Shortest job first can be either preemptive or non-preemptive. Owing to its simple nature, shortest job first is considered optimal. It also reduces the average waiting time for other processes awaiting execution.

Shortest job first is also known as shortest job next (SJN) and shortest process next (SPN).

Shortest job first depends on the average running time of the processes. The accurate estimates of these measures help in the implementation of the shortest job first in an environment, which otherwise makes the same nearly impossible to implement. This is because often the execution burst of processes does not happen beforehand. It can be used in interactive environments where past patterns are available to determine the average time between the waiting time and the commands. Although it is disadvantageous to use the shortest-job-first concept in short-term CPU scheduling, it is considered highly advantageous in long-term CPU scheduling. Moreover, the throughput is high in the case of shortest job first.

Shortest job first also has its share of disadvantages. For one, it can cause process starvation for longer jobs if there are a large number of shorter processes. Another is the need to know the execution time for each process beforehand. Often, this is almost impossible in many environments.

**Main Algorithm used :**

SET I = 0;

while (i < condition)

{

for (let i = 0; i < 3; i++)

{

If(condition)

{

Swap;

}

i++

}

SET I = 0;

while (i < condition) {

ToTaL += BUR\_time[i-1]

WAIT\_time[i] = ToTaL - ARI\_time[i]

Final\_WAit += WAIT\_time[i]

i++

}

SET I = 0;

while (i < condition) {

ToTaL\_2 += BUR\_time[i]

T\_ARD\_time[i] = ToTaL\_2 - ARI\_time[i]

Final\_Turn\_ARND += T\_ARD\_time[i]

i++

}

**Overall complexity :** O(n^2)

**Constraints given in the problem :**

**-** The priority of each job is dependent on its estimated run time, and also the amount of time it has spent waiting.

- The jobs that have spent a long time waiting compete against those estimated to have short run times.

- Priority = 1+ Waiting time / Estimated run time

**Boundary condition :**

* **Process arriving first will always be executed first thoroughly as per the non-preemptive scheduling concept.**

**Test cases applied :**



**GitHub link :** https://github.com/klaus1002/OS\_CA\_Project.git