Operating Systems

CSE – 316

Name – Maries Tahaab

Student Id – 11701099

Section – EE032 A01

E-Mail – mariestahaab@gmail.com

GitHub - <https://github.com/mariestahaab/Proj_OS>

*Question 1:*

*Develop a scheduler which submits the processes to the processor in the following scenario, and compute the scheduler performance by providing the waiting time for process, turnaround time for process and average waiting time and turnaround time. Considering the arrival time and the burst time requirement of the processes the scheduler schedules the processes by interrupting the processor after every 3 units of time and does consider the completion of the process in this iteration. The schedulers than checks for the number of processes waiting for the processor and allots the processor to the process but interrupting the processor after every 6 units of time and considers the completion of the process in this iteration. The scheduler after the second iteration checks for the number of processes waiting for the processor and now provides the processor to the process with the least time requirement to go in the terminated state.*

Explanation:

* The process with the shortest burst time will run for the first 3 units of time.

The type of algorithm used will be *SJF Pre-emptive*.

* After the pre-emption of the previous process, the process with the shortest burst time executes for the next 6 units of time.

The type of algorithm used will be *SJF Pre-emptive*.

* The execution of the previous process is stopped and the process with the shortest burst time executes till completion.

The type of algorithm used will be *SJF Non-Pre-emptive*.

**CODE:**

#include<stdio.h>

int n **=** 4**,** units**=**0**;**

float a\_wt **=** 0**,** a\_tat **=** 0**;**

int at**[]** **=** **{**0**,** 2**,** 4**,** 13**};** //needs to be in ascending order

int bt**[]** **=** **{**18**,** 23**,** 13**,** 10**};**

int bt\_i**[]** **=** **{**18**,** 23**,** 13**,** 10**};**

int ct**[]** **=** **{**0**,** 0**,** 0**,** 0**};**

int tat**[]** **=** **{**0**,** 0**,** 0**,** 0**};**

int wt**[]** **=** **{**0**,** 0**,** 0**,** 0**};**

void main**()**

**{**

int i**,** temp**[**n**],** pcs**;**

units **=** at**[**smallest**(**at**)];**

//First Three Units: SJF Pre-emptive

pcs **=** smallest**(**at**);**

bt**[**pcs**]=**bt**[**pcs**]-**3**;**

units **=** units**+**3**;**

//Next Six units: SJF Pre-emptive

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

temp**[**i**]** **=** 0**;**

**if(**at**[**i**]<=**units**){**

temp**[**i**]** **=** at**[**i**];**

**}**

**}**

pcs **=** smallest\_lim**(**bt**,** greatest**(**temp**));**

bt**[**pcs**]** **=** bt**[**pcs**]-**6**;**

units **=** units**+**6**;**

//SJF Non Pre-emptive

**while(**incomplete**(**bt**)){**

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

temp**[**i**]** **=** 0**;**

**if(**at**[**i**]<=**units**){**

temp**[**i**]** **=** at**[**i**];**

**}**

**}**

pcs **=** smallest\_lim**(**bt**,** greatest**(**temp**));**

units **=** units **+** bt**[**pcs**];**

bt**[**pcs**]** **=** 0**;** //bt[pcs] = bt[pcs]-bt[pcs];

ct**[**pcs**]** **=** units**;**

**}**

//Calculating TAT, WT, A\_TAT, A\_WT

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

tat**[**i**]=**ct**[**i**]-**at**[**i**];**

a\_tat **=** a\_tat **+** tat**[**i**];**

wt**[**i**]=**tat**[**i**]-**bt\_i**[**i**];**

a\_wt **=** a\_wt **+** wt**[**i**];**

**}**

a\_tat **=** a\_tat**/**n**;**

a\_wt **=** a\_wt**/**n**;**

//Displaying Results

printf**(**"\nProcess\t ArrivalTime\tBurstTime\tCompletionTime\t TurnAroundTime\t WaitingTime\n"**);**

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

printf**(**"\nP%d"**,** i**);**

printf**(**"\t %d"**,** at**[**i**]);**

printf**(**"\t\t%d"**,** bt\_i**[**i**]);**

printf**(**"\t\t%d"**,** ct**[**i**]);**

printf**(**"\t\t %d"**,** tat**[**i**]);**

printf**(**"\t\t %d"**,** wt**[**i**]);**

**}**

printf**(**"\n\nAverage Turn Around Time - %f"**,**a\_tat**);**

printf**(**"\nAverage Waiting Time - %f"**,** a\_wt**);**

**}**

//Function Declaration

int greatest**(**int x**[]){**

int i**,** index**=**0**,** greatest**=**x**[**0**];**

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

**if(**x**[**i**]>**greatest**){**

greatest **=** x**[**i**];**

index **=** i**;**

**}**

**}**

**return** index**;**

**}**

int smallest**(**int x**[]){**

int i**,** index**=**0**,** smallest**=**x**[**0**];**

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

**if(**x**[**i**]<**smallest**){**

smallest **=** x**[**i**];**

index **=** i**;**

**}**

**}**

**return** index**;**

**}**

int smallest\_lim**(**int x**[],** int limit**){**

int i**,** index**=**0**,** smallest**,** temp**[**n**];**

**for(**i**=**0**;** i**<=**limit**;** i**++){**

temp**[**i**]** **=** x**[**i**];**

**}**

smallest **=** temp**[**greatest**(**temp**)];**

**for(**i**=**0**;** i**<=**limit**;** i**++){**

**if(**x**[**i**]** **!=** 0**){**

**if(**x**[**i**]<=**smallest**){**

smallest **=** x**[**i**];**

index **=** i**;**

**}**

**}**

**}**

**return** index**;**

**}**

int incomplete**(**int x**[]){**

int i**,** result **=** x**[**0**];**

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

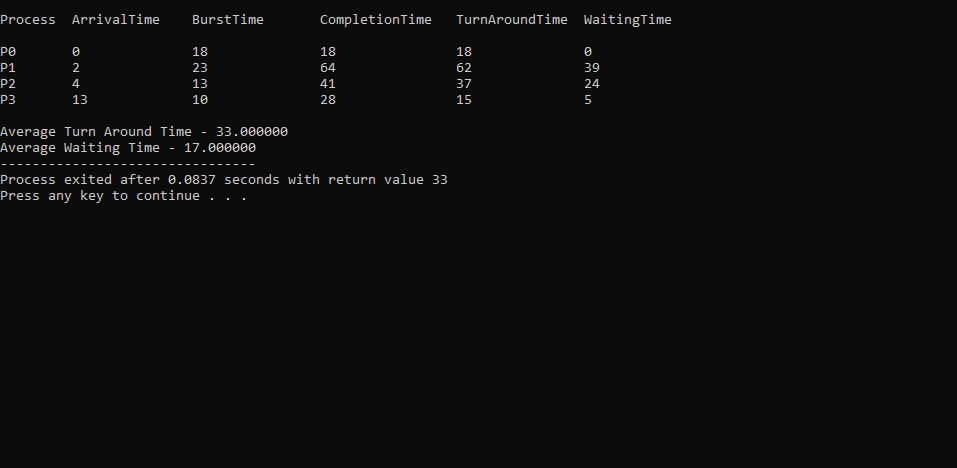
result **=** result**||**x**[**i**];**

**}**

**return** result**;**

**}**

**Output:**

****

**Functions Used:**

* ***greatest ()***: This function takes an integer array as argument and returns the index of the element having the highest value.
* ***smallest ()***: This function takes an integer array as argument and returns the index of the element having the smallest value. *It is important to note that this function considers 0 as the smallest value*.
* ***smallest\_lim ()***: This function takes an integer array and an integer variable *(limit)* as arguments and returns the index of the element having the smallest value. The array is not traversed completely, but only up to the *limit* variable, which contains the index value up to which the array is to be traversed. *It is important to note that this function considers numbers other than 0 as the smallest value*.
* ***incomplete ()***: This function takes an integer array as argument and returns integer values 1 or 0, depending on the contents of the array. If all the elements in the array are 0, the value returned is 0. On the other hand, if either of the elements in the array has a value other than 0, the value returned is 1.