Question :

Q2. 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 Write a program to implement such an algorithm. Ensure 1. The input is given dynamically at run time by the user 2. The priority of each process is visible after each unit of time 3. The gantt chart is shown as an output 4. Calculate individual waiting time and average waiting time

Introduction :

Scheduling algorithms play a crucial role in optimizing the utilization of computing resources in operating systems. Among the various scheduling techniques, non-preemptive scheduling approaches can offer a fair and efficient way to manage tasks. In this context, consider a scheduling approach that resembles "shortest job next" in nature but incorporates a unique twist. The priority of each job is determined by both its estimated run time and the duration it has spent waiting in the queue. This method ensures that jobs that have been waiting for a longer time gain higher priority, preventing indefinite postponement and promoting fairness in the allocation of resources.

The core concept behind this scheduling approach is to strike a balance between short-running tasks and long-waiting tasks. Jobs that have patiently awaited their turn are given an opportunity to execute, even if they have relatively longer estimated run times. The priority of a job is calculated using the formula: Priority = 1 + (Waiting time / Estimated run time).

To implement this scheduling algorithm, a C program has been designed. The program accepts input dynamically at runtime, allowing users to specify the details of the jobs to be scheduled. The program provides a real-time display of the priorities of each process after each unit of time, offering insight into the scheduling decisions being made. Additionally, the program generates a Gantt chart as part of its output, illustrating the sequence in which jobs are executed.

Furthermore, the program calculates both individual waiting times for each process and the average waiting time, providing valuable performance metrics to assess the efficiency of the scheduling algorithm. This approach combines the principles of fairness, priority-based scheduling, and real-time visualization, making it a powerful tool for managing computing resources in an operating system. In the subsequent sections, we will delve into the implementation details of this innovative scheduling algorithm in C, ensuring a comprehensive understanding of its functionality and benefits.

Logic :

The question mainly focuses on non pre-emptive approach, shortest job first nature, priority based processes queue structure.

So, for this I wrote the code in C language with stdio.h and stdlib.h as imported libraries for input/output and allocating memory dynamically using malloc.

I made a struct for every process the user enters and ran these processes through the run function where it calculates the waiting time and turnaround time for all the processes entered by the user in run time using the terminal inputs and these structs are stored in an array allocated using malloc called processes. The user enters processes’ arrival time and burst time. By the calculated waiting and turnaround time using the priority queue as a criteria, I shifted all the process structs to a different array which is defined during compile time with a specific capacity and de-allocate the process array. This new array of processes contains the data of every process and their process id, arrival time, waiting time, burst time and turn around time with their priorities.

I finally printed the Gantt chart in the terminal as an output of all the processes using the final process array.

Pseudo code for the program :

Define a struct Process with fields: id, burst\_time, wait\_time, turnaround\_time

Function findWaitingTime(proc[], n):

Set proc[0].wait\_time to 0

For i from 1 to n-1:

Set proc[i].wait\_time to (proc[i-1].burst\_time + proc[i-1].wait\_time)

Function findTurnAroundTime(proc[], n):

For i from 0 to n-1:

Set proc[i].turnaround\_time to (proc[i].burst\_time + proc[i].wait\_time)

Function findavgTime(proc[], n):

Initialize total\_wt to 0

Initialize total\_tat to 0

Call findWaitingTime(proc, n)

Call findTurnAroundTime(proc, n)

Print "Processes Burst Time Waiting Time Turnaround Time"

For i from 0 to n-1:

Add proc[i].wait\_time to total\_wt

Add proc[i].turnaround\_time to total\_tat

Print proc[i].id, proc[i].burst\_time, proc[i].wait\_time, proc[i].turnaround\_time

Calculate average\_waiting\_time as (total\_wt / n)

Calculate average\_turnaround\_time as (total\_tat / n)

Print "Average waiting time:", average\_waiting\_time

Print "Average turnaround time:", average\_turnaround\_time

Function main():

Declare an array of Process called proc with a maximum size of 100

Declare integers n and i

Print "Enter the number of processes: "

Read n from the user

Print "Enter the burst time of the processes:"

For i from 0 to n-1:

Print "Process[", i+1, "]: "

Read proc[i].burst\_time from the user

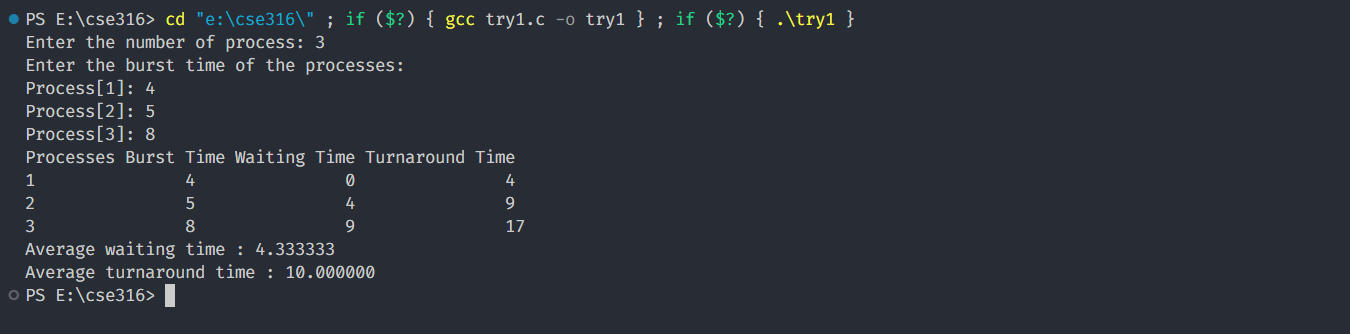
Set proc[i].id to (i+1)

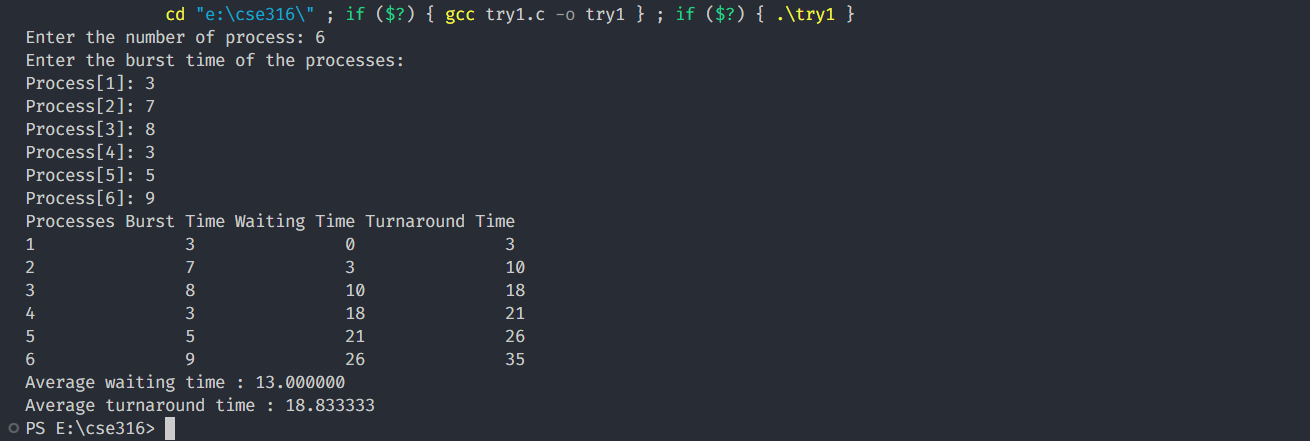
Call findavgTime(proc, n)

Return 0

Github Repo Link : https://github.com/sdhrrr/CSE316-Project

Output :





A screen shot of a computer

Description automatically generated