**Student Name : Dipankar Lama**

**Student ID : 11803378**

**Email Address :** [**lamadipankar35@gmail.com**](mailto:lamadipankar35@gmail.com)

**GitHub Link :** [**https://github.com/Antiru/Operating-System-Assesment.git**](https://github.com/Antiru/Operating-System-Assesment.git)

**Code**

#include <bits/stdc++.h>

using namespace std;

struct process {

int at ;

int bt ;

int pr ;

int pno;

};

process proc[100];

int no\_of\_processes ;

/\*

Writing comparator function to sort according to priority if

arrival time is same

\*/

bool compare\_condition(process a,process b) {

if(a.at == b.at) {

return a.pr < b.pr;

}

else

{

return a.at < b.at;

}

}

// Using FCFS Algorithm to find Waiting time

void getWaitingTime(int wt[])

{

// declaring service array that stores cumulative burst time

int service[50] ;

// Initilising initial elements of the arrays

service[0] = 0 ;

wt[0] = 0 ;

for(int i = 1 ; i < no\_of\_processes ; i++)

{

service[i] = proc[i-1].bt + service[i-1] ;

wt[i] = service[i] - proc[i].at + 1 ;

// If waiting time is negative, change it into zero

if(wt[i] < 0) {

wt[i] = 0 ;

}

}

}

void getTurnaroundTime(int tat[],int wt[])

{

// Filling turnaroundtime array

for(int i = 0 ; i < no\_of\_processes ; i++)

{

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

}

}

void GanttChart()

{

//Declare waiting time and turnaround time array

int wt[50], tat[50] ;

double wavg=0, tavg=0;

// Function call to find waiting time array

getWaitingTime(wt);

//Function call to find turnaround time

getTurnaroundTime(tat, wt);

int stime[50], ctime[50];

stime[0] = 1 ;

ctime[0] = stime[0] + tat[0] ;

// calculating starting and ending time

for(int i = 1 ; i < no\_of\_processes ; i++)

{

stime[i] = ctime[i-1] ;

ctime[i] = stime[i] + tat[i] - wt[i] ;

}

cout << "[+]------------------------------------------------------------------------------------[+]\n" ;

cout << "[+] Process\_no | Start\_time | Complete\_time | Turn\_Around\_Time | Waiting\_Time [+]"<<endl;

// display the process details

for(int i = 0 ; i < no\_of\_processes ; i++)

{

wavg += wt[i] ;

tavg += tat[i] ;

cout << "[+]\t " << proc[i].pno << "\t |\t" << stime[i] << "\t|\t" << ctime[i] << "\t |\t " << tat[i] << "\t |\t" << wt[i] <<" [+]"<< endl;

}

cout << "[+]------------------------------------------------------------------------------------[+]\n" ;

// display the average waiting time

//and average turn around time

cout << endl ;

cout << "[-] Average waiting time is : ";

cout << wavg/(float)no\_of\_processes<<endl;

cout << "[-] Average turnaround time : ";

cout << tavg/(float)no\_of\_processes<<endl;

}

int main()

{

// Color Coding

char blue[] = { 0x1b, '[', '1', ';', '3', '4', 'm', 0 };

char normal[] = { 0x1b, '[', '0', ';', '3', '9', 'm', 0 };

//

int arrivaltime[100] ;

int bursttime[100] ;

int priority[100] ;

cout << blue << "[+]-----------------------------------------------------------------------[+]\n" ;

cout << "[+] Priority Scheduling Algorithm [+]\n" ;

cout << "[+] Dipankar Lama [+]\n" ;

cout << "[+] C++ [+]\n" ;

cout << "[+]-----------------------------------------------------------------------[+]\n" << normal ;

cout << endl ;

cout << "[-] Enter The Number of Processes : " ;

cin >> no\_of\_processes ;

cout << endl ;

for(int i = 0 ; i < no\_of\_processes ; i++) {

int inputFromUser ;

cout << "[+] Enter Process Number : " ;

cin >> inputFromUser ;

cout << "[+] Enter The Priority of the Process : " ;

cin >> priority[i] ;

//new\_process -> setPriority(inputFromUser) ;

cout << "[+] Enter The Burst Time of the Process : " ;

cin >> bursttime[i] ;

//new\_process -> setBurstTime(inputFromUser) ;

cout << "[+] Enter The Arrival Time of the Process : " ;

cin >> arrivaltime[i] ;

//new\_process -> setArrivalTime(inputFromUser) ;

cout << endl ;

}

for(int i = 0 ; i < no\_of\_processes ; i++)

{

proc[i].at = arrivaltime[i] ;

proc[i].bt = bursttime[i] ;

proc[i].pr = priority[i] ;

proc[i].pno = i + 1 ;

}

sort(proc, proc + no\_of\_processes, compare\_condition) ; // Sorting using inbuild Sort function in STL

GanttChart() ; //Calling function findgc for finding Gantt Chart

return 0 ;

}

**Description**

Priority scheduling is a method of scheduling processes based on priority. In this method, the scheduler chooses the tasks to work as per the priority, which is different from other types of scheduling, for example, a simple round robin.

Priority scheduling involves priority assignment to every process, and processes with higher priorities are carried out first, whereas tasks with equal priorities are carried out on a first-come-first-served (FCFS) or round robin basis. An example of a general-priority-scheduling algorithm is the shortest-job-first (SJF) algorithm.

Priority scheduling can be either of the following:

* Preemptive: This type of scheduling may preempt the central processing unit (CPU) in the case the priority of the freshly arrived process being greater than those of the existing processes.
* Non-preemptive: This type of scheduling algorithm simply places the new process at the top of the ready queue.

Indefinite blocking, otherwise called starvation, is one of the major issues concerning priority scheduling algorithms. It is a state where a process is ready to be executed, but faces a long wait in getting assigned to the CPU.

It is often possible that a priority scheduling algorithm can make a low-priority process wait indefinitely. For example, in an intensely loaded system, if there are a number of higher priority processes, the low-priority processes may never get the CPU for execution.

A remedy to starvation is aging, which is a technique used to gradually increase the priority of those processes that wait for long periods in the system.

**Algorithm**

1. First input the processes with their arrival time, burst time and priority.
2. Sort the processes, according to arrival time if two process arrival time is same then sort according process priority if two process priority are same then sort according to process number.
3. Now simply apply FCFS algorithm.
4. Prepare a Gantt Chart
5. Calculate Average Waiting Time and Turnaround Time
6. Display The result

**Complexity**

|  |  |
| --- | --- |
| Line Number | Complexity & description |
| 1-2 | Importing Module O(1) |
| 5-10 | Creating a user-define variable to store process. O(1) |
| 19-28 | Used as a utility for sorting the array of process type O(1) |
| 31-54 | This entire section is used to calculate waiting time of each process. O(no\_of\_processes) i.e O(n) |
| 56-65 | This function calculates the TurnAround Time of every process. O(no\_of\_processes) i.e O(n) |
| 67-111 | Prepare a Gantt Chart for furthur calculation and displays the output in the form of table. O(no\_of\_processes) i.e O(n) |
| 113-173 | Main Function which takes input from the user and also Sort the array of process type using inbuild STL and call GanttChart for further process. |

**The Overall Complexity of the Program is O(n).**

**Constrains**

**1.**Indefinite blocking or starvation.

**2.**A priority scheduling can leave some low priority waiting processes indefinitely for CPU.

**3.**If the system eventually crashes then all unfinished low priority processes gets lost

**Test Cases:**

|  |  |  |  |
| --- | --- | --- | --- |
| Process | Priority | Burst Time | Arrival Time |
| P1 | 1 | 4 | 0 |
| P2 | 2 | 3 | 0 |
| P3 | 1 | 7 | 6 |
| P4 | 3 | 4 | 11 |
| P5 | 2 | 2 | 12 |

