Name: Aikansh Choudhary

Section: K17HU

Roll. No: 51

Registration No: 1705215

**Introduction:**

Priority scheduling is a non-pre-emptive algorithm and one of the most common scheduling algorithms in batch systems. Each process is assigned first arrival time (less arrival time process first) if two processes have same arrival time, then compare to priorities (highest process first). Also, if two processes have same priority then compare to process number (less process number first). This process is repeated while all process get executed.

**Algorithm:**

In general, a multilevel feedback queue scheduler is defined by the following parameters:

* The number of queues.
* The scheduling algorithm for each queue.
* The method used to determine when to upgrade a process to a higher-priority queue.
* The method used to determine when to demote a process to a lower-priority queue.
* The method used to determine which queue a process will enter when that process needs service.

The definition of a multilevel feedback queue scheduler makes it the most general CPU-scheduling algorithm. It can be configured to match a specific system under design. Unfortunately, it also requires some means of selecting values for all the parameters to define the best scheduler. Although a multilevel feedback queue is the **most general scheme**, it is also the **most complex**.

**Description:**

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.

CODE:

#include <bits/stdc++.h>

using namespace std;

#define totalprocess 5

// Making a struct to hold the given input

struct process

{

int at,bt,pr,pno;

};

process proc[50];

/\*

Writing comparator function to sort according to priority if

arrival time is same

\*/

bool comp(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 get\_wt\_time(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<totalprocess;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 get\_tat\_time(int tat[],int wt[])

{

// Filling turnaroundtime array

for(int i=0;i<totalprocess;i++)

{

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

}

}

void findgc()

{

//Declare waiting time and turnaround time array

int wt[50],tat[50];

double wavg=0,tavg=0;

// Function call to find waiting time array

get\_wt\_time(wt);

//Function call to find turnaround time

get\_tat\_time(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<totalprocess;i++)

{

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

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

}

cout<<"Process\_no\tStart\_time\tComplete\_time\tTurn\_Around\_Time\tWaiting\_Time"<<endl;

// display the process details

for(int i=0;i<totalprocess;i++)

{

wavg += wt[i];

tavg += tat[i];

cout<<proc[i].pno<<"\t\t"<<

stime[i]<<"\t\t"<<ctime[i]<<"\t\t"<<

tat[i]<<"\t\t\t"<<wt[i]<<endl;

}

// display the average waiting time

//and average turn around time

cout<<"Average waiting time is : ";

cout<<wavg/(float)totalprocess<<endl;

cout<<"average turnaround time : ";

cout<<tavg/(float)totalprocess<<endl;

}

int main()

{

int arrivaltime[] = { 1, 2, 3, 4, 5 };

int bursttime[] = { 3, 5, 1, 7, 4 };

int priority[] = { 3, 4, 1, 7, 8 };

for(int i=0;i<totalprocess;i++)

{

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

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

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

proc[i].pno=i+1;

}

//Using inbuilt sort function

sort(proc,proc+totalprocess,comp);

//Calling function findgc for finding Gantt Chart

findgc();

return 0;

}