OPERATING SYSTEMS LAB PROJECT

Instructor: Ashu Abdul

Group Members:

E. Sravanthi AP19110010354
K. Padmini AP19110010381
A. Nanda Kishore AP19110010391
P. Satya Akash AP19110010393
K. Bineeth Kumar AP19110010396

Develop/Simulate a preemptive scheduling algorithm

Problem Statement:

Currently prevalent round robin preemptive scheduling algorithms are not applicable in real time operating systems because of their large waiting time, large turnaround time, large response time, high context switch rates and less throughput. The primary aim of this project is to come up with a method for round robin scheduling which enhances the CPU performance in a real time operating system. The proposed algorithm for real time systems is a hybrid of priority and round-robin scheduling algorithms. It combines the merits of round robin, like elimination of starvation, and priority scheduling. The proposed algorithm also deals with aging by assigning variable priorities to the processes. Thus, the algorithm corrects all the limitations of round robin scheduling algorithm. We will also present a comparative analysis of our new algorithm with the existing algorithm i.e., round robin based on average turnaround time, average waiting time, varying time quantum and number of context switches.

Round Robin Scheduling Algorithm:

The round robin algorithm was designed mainly for time-shared systems not for real time systems. Time slice or time quantum is defined in case of RR algorithm, which refers to duration for which the process is allocated to the CPU and executed.

The processes which have to be executed are kept in a circular queue which has a head and a tail. The CPU scheduler will go around the queue, allocating the CPU to each process for a time interval of one quantum but the problem is that all the processes are arranged in FCFS (First Come First Serve) manner.

Arriving processes are then added to the tail of the queue.

The CPU scheduler will then select the Process Control Block from the head of the ready queue. This is a disadvantage in RR algorithm since all processes are basically given the same priority. Round robin also favours the process with short CPU burst and penalizes long ones.

Disadvantages:

The disadvantages of round robin CPU scheduling algorithm which affects execution process time shared system are as follows

1. Larger waiting, response time and high rates of context switching. Since Real-time programs must guarantee response within specified time constraints therefore larger waiting, response time and high rates of context switching affect the system's performance and delay the results.

2. Low throughput (Throughput is defined as number of processes completed per time unit). If round robin is implemented in soft real time systems throughput will be low which leads to severe degradation of system performance because of high context switching. If the number

of context switches is low, then the throughput will be high. Context switch and throughput are inversely proportional to each other. With these observations it is found that the existing simple round robin architecture is not suitable for real time

Algorithm:

We calculate an Intelligent Time Slice (ITS) is calculated based on the priority, shortest CPU burst time and context switch. We have a pre-defined time slice which is the Original Time Slice (OTS), if the process needs special consideration, the Priority Component (PC) is assigned to 0 or 1 according to the pre-defined priority by the user. We define a Shortness Component (SC) which is the difference between the burst time of the current process and its previous process. If the difference is negative, we make SC as 1 else 0. To calculate Context Switch Component (CSC) we add PC, SC and OTP and the resulting sum is subtracted from the burst time of the process. If the result obtained is less than the OTS, we consider it as CSC. Adding all the calculated components we get our ITS.

1. Sort the processes according to priority as well as shortness and assign a new priority to each process which is the sum of the original priority and shortness rank.

2. Calculate priority component. If there are n processes, for a process i in n
   PCi=0 if its new priority is > 2n/3 (Not Important)
   PCi=1 if its new priority is > n/3 (Moderately Important)
   PCi=2 if its new priority is >= 1 (Important)

3. Calculate shortness component. If there are n processes, for a process i in n
   SCi=0 if the (Burst Time)i>(Burst Time)i-1 (Longer)
   SCi=1 if the (Burst Time)i<=(Burst Time)i-1 (Shorter)

4. Calculate the intelligent time slice (ITS) for each process as the sum of the initial time slice, burst time, priority component and shortness component

5. Repeat Step 6 until all processes are completed.

6. If the Round number(j) is 1, calculate time quantum as
   TQj,i=ITSi if SCi=1
   TQj,i=ITSi/2 if SCi=0
   If the Round number(j) is not 1, calculate time quantum as
   TQj,i=TQj-1,i * 2 if SCi=1
   TQj,i=TQj-1,i * 1.5 if SCi=0

7. Calculate average waiting time and average turnaround time.

Code:

```
#include <iostream>
#include <math.h> //for mathematical calculation
#include <iomanip> //header file used of setw() function
using namespace std;
int main()
{
  int n;
  cout << "\nEnter the number of processes: ";
  cin >> n; //processes size
  int bt[n], p[n], s[n], wt[n], tat[n], ts, its[n], tq[n][n], rbt[n], ord[n]; //bt=burst time,
p=priority, s=shortness component, tat=turn around time, its=intelligent time slice, tq=time
quantum
  //rbt=remaining burst time, ord=order

//initialization of waiting time, turn around time, shortness component and time quantum
  for (int i = 0; i < n; i++)
  {
    wt[i] = tat[i] = 0;
    s[i] = 1;
    for (int j = 0; j < n; j++)
      tq[i][j] = 0;
  }

  cout << "\nEnter the initial time slice: ";
  cin >> ts; //input time first time slice

  for (int i = 0; i < n; i++)
  {
    ord[i] = i + 1; //increment the order by 1 for each iteration
```

```
  cout << "\nEnter burst time for process " << i + 1 << ": ";
  cin >> bt[i]; //take burst time for each process
  cout << " \nEnter the priority of the process " << i + 1 << ": ";
  cin >> p[i]; //enter the priority of each process
}

int flag = 0, j = 0;
//Sorting the process according to their burst times

for (int i = 0; i < n - 1; i++)
  for (int j = 0; j < n - 1; j++)
    if (bt[j] > bt[j - 1])
    {
      //swapping burst time based on burst time
      int t = bt[j];
      bt[j] = bt[j + 1];
      bt[j + 1] = t;
      //swapping prioirity based on burst time
      t = p[j];
      p[j] = p[j + 1];
      p[j + 1] = t;
      //swapping order based on burst time
      t = ord[j];
      ord[j] = ord[j + 1];
      ord[j + 1] = t;
    }

for (int i = 0; i < n; i++)
  p[i] += i; //priority incrementing
```

```
//Sorting processes according to priorities
for (int i = 0; i < n - 1; i++)
  for (int j = 0; j < n - 1; j++)
    if (p[j] > p[j + 1]) //if the priority of current process is more than next process's priority
    {
      //swapping burst times
      int t = bt[j];
      bt[j] = bt[j + 1];
      bt[j + 1] = t;
      //swapping priorities
      t = p[j];
      p[j] = p[j + 1];
      p[j + 1] = t;
      //swapping order
      t = ord[j];
      ord[j] = ord[j + 1];
      ord[j + 1] = t;
    }

// at the initial stage as the processing wasn't started the burst time
// be the same as remaining burst times
for (int i = 0; i < n; i++)
  rbt[i] = bt[i]; //rbt=remaining burst time

while (!flag) //flag is working as an indicator
{
  for (int i = 0; i < n; i++)
  {
     //prioritizing each component
    if (p[i] > 0.67 * n) //(2n/3)
```

```
      p[i] = 0;
    else if (p[i] > 0.33 * n) //(n/3)
      p[i] = 1;
    else
      p[i] = 2;

    if (i != 0)
      if ((bt[i] - bt[i - 1]) > 0) //defining shortness component based on burst time of current and previous process
        s[i] = 0;
   //calculating intelligent time slice of each process
    its[i] = ts + bt[i] + s[i] + p[i];

//*********************************************************************//
    if (j == 0) //setting time quantum and remaining burst time of round (1)
    {
      if (s[i] == 1)
        tq[j][i] = its[i];
      else
        tq[j][i] = ceil(0.5 * (float)its[i]); //ceil(x) : Returns the smallest integer that is greater than or equal to x (i.e : rounds up the nearest integer).

      if (rbt[i] < tq[j][i])
        tq[j][i] = rbt[i];


      rbt[i] = rbt[i] - tq[j][i];

    }
    // rounds are represented along x-axis
    // time quantums for each process is represented along y-axis
```

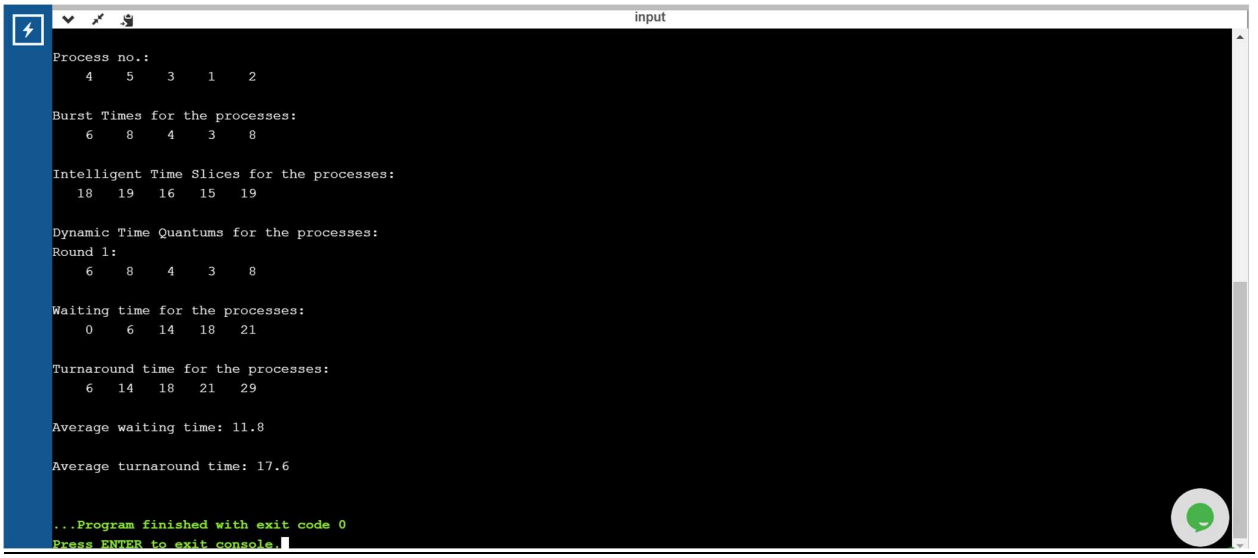
```
    else //setting time quantum and remaining burst time for remaining rounds
    {
      if (rbt[i] <= 0)
        tq[j][i] = 0;
      else if (s[i] == 1)
        tq[j][i] = 2 * tq[j - 1][i];
      else
        tq[j][i] = 1.5 * tq[j - 1][i];

      if (rbt[i] < tq[j][i])
        tq[j][i] = rbt[i];

      rbt[i] = rbt[i] - tq[j][i];
    }
  }


  j++; //j is for going to next value in a round(y-axis), i is for rounds iteration(x-axis)

  flag = -1;
  for (int i = 0; i < n; i++)
    if (rbt[i] > 0) //breaking while loop
      flag = 0;
 }
//*************************************************************//

 //print outputs
 cout << "\n\nProcess no.:\n";
 for (int i = 0; i < n; i++)
```

```
  cout << setw(5) << ord[i]; //setw(5) is a function which gives space of 5 in output

cout << "\n\nBurst Times for the processes:\n";
for (int i = 0; i < n; i++)
  cout << setw(5) << bt[i];

cout << "\n\nIntelligent Time Slices for the processes:\n";
for (int i = 0; i < n; i++)
  cout << setw(5) << its[i];

cout << "\n\nDynamic Time Quantums for the processes:\n";
for (int x = 0; x < j; x++)
{
  cout << "Round " << x + 1 << ":" << endl;
  for (int y = 0; y < n; y++)
    cout << setw(5) << tq[x][y];
  cout << endl;
}

//calculation of wait time and turn around time
for (int x = 0; x < n; x++)
{
  flag = -1;
  for (int y = 0; y < j; y++)
  {
    for (int z = 0; z < n; z++)
    {
      if (z != x)
        wt[x] += tq[y][z];
      else if (z == x && tq[y + 1][z] == 0)
```

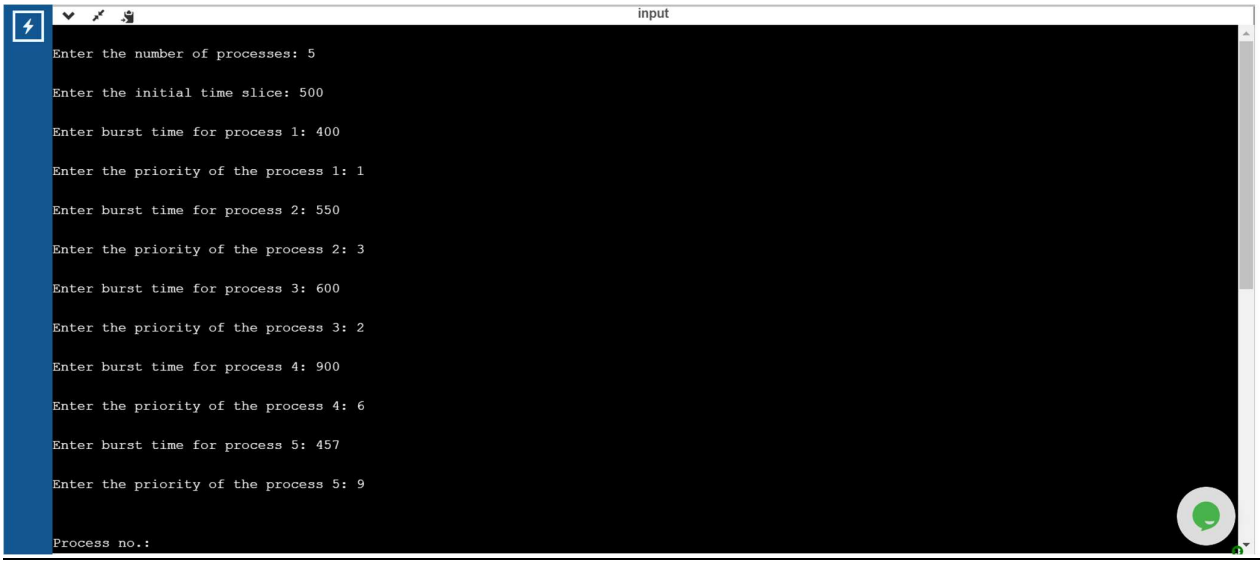
```
        {
          flag = 0;
          break;
        }
      }
      tat[x] += tq[y][x];
      if (flag == 0)
        break;
    }
    tat[x] += wt[x];
  }
  //printing WT and TAT
  cout << "\nWaiting time for the processes:\n";
  for (int i = 0; i < n; i++)
    cout << setw(5) << wt[i];
  cout << "\n\nTurnaround time for the processes:\n";
  for (int i = 0; i < n; i++)
    cout << setw(5) << tat[i];
//calculate avg wait time and turn around time
  float avwt = 0, avtat = 0;
  for (int i = 0; i < n; i++)
  {
    avwt += wt[i];
    avtat += tat[i];
  }
  avwt /= n;
  avtat /= n;
  cout << "\n\nAverage waiting time: " << avwt << endl;
  cout << "\nAverage turnaround time: " << avtat << endl;
}
```

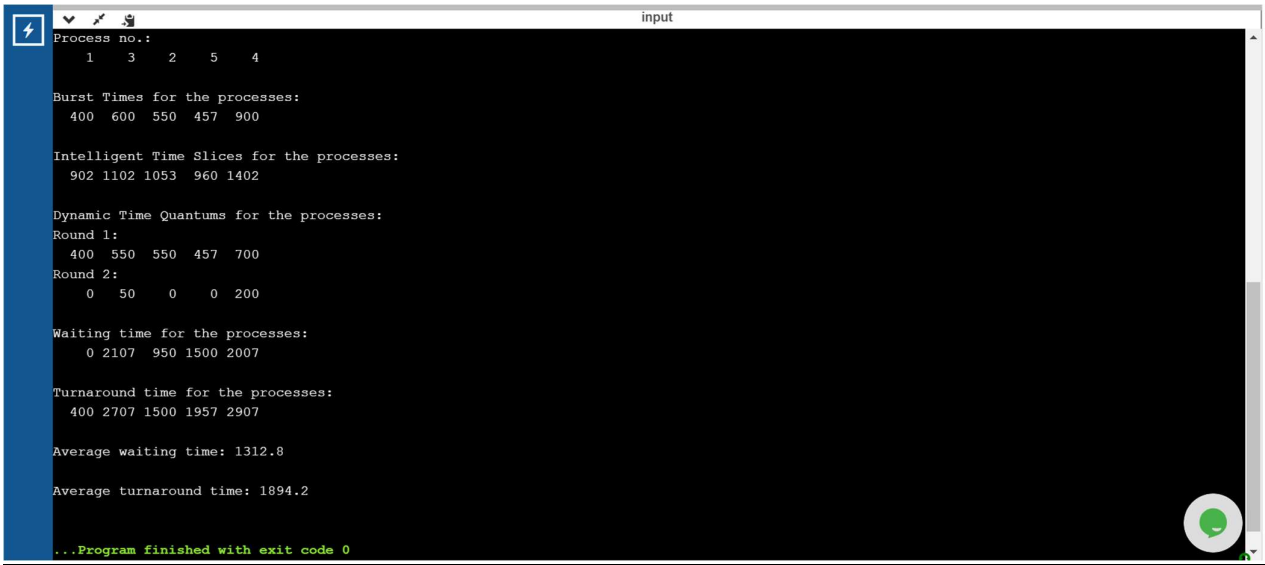
Output:

Low burst time:



High burst time:





Graphical Representation of AWT and TAT:

The proposed algorithm has less Avg. waiting time and Avg. turnaround time as compared to RR and Priority Algorithm.

Github Link: https://github.com/bineethkumar/Preemptive-Scheduling