Lab 5. Scheduling CPU Algorithms

Yêu cầu: Cho n tiến trình (Process) với thời điểm vào RQ (Arrival Time), thời gian sử dụng CPU (Burst Time) và độ ưu tiên của các tiến trình. Viết chương trình điều phối CPU cho các tiến trình, tính và in ra màn hình thời gian đợi trung bình (average waiting time), thời gian hoàn tất trung bình (average turn around time) của hệ thống và thứ tự điều phối của các tiến trình. Sử dụng các thuật toán:

- ✓ **FIFO**
- ✓ **Round Robin**
- ✓ **Priority Scheduling (Độc quyền/Không độc quyền)**
- ✓ **Short Job First (Độc quyền/Không độc quyền)**

Completion Time: thời điểm tiến trình kết thúc
Around Time = Completion Time – Arrival Time
Waiting Time = Turn Around Time – Burst Time

I. Điều phối CPU theo thuật toán FIFO (FCFS)

Input:

bt[] : là burst time của tất các các tiến trình.

at[]: là arrival time của các tiến trình

wt[]: là waiting time của các tiến trình.

tat[]: là Turn Around Time của các tiến trình

Nhận xét:

- ✓ **Thứ tự điều phối của các tiến trình là thứ tự vào của các tiến trình.**
- ✓ **Để tiến trình i thực thi, thì (i-1) tiến trình trước đó phải kết thúc nên ta có:**

wt[i] = (bt[0] + bt[1] +…… bt[i-1]) – at[i]

tat[i] = wt[i] + bt[i]

Codes C#

```
namespace FIFOScheduling
```

```
{
   public class FIFO
   {
      // Function to find the waiting time for all
      // processes
      static void findWaitingTime(int[] processes, int n, int[] bt, int[] wt, int[] at)
      {
         int[] service_time = new int[n];
         service_time[0] = 0;
         wt[0] = 0;

         // calculating waiting time
         for (int i = 1; i < n; i++)
         {
            // Add burst time of previous processes
            service_time[i] = service_time[i - 1] + bt[i - 1];

            // Find waiting time for current process =
            // sum - at[i]
            wt[i] = service_time[i] - at[i];

            // If waiting time for a process is in negative
            // that means it is already in the ready queue
            // before CPU becomes idle so its waiting time is 0
            if (wt[i] < 0)
               wt[i] = 0;
         }
      }
      // Function to calculate turn around time
      static void findTurnAroundTime(int[] processes, int n, int[] bt,
                                     int[] wt, int[] tat)
      {
         // Calculating turnaround time by adding bt[i] + wt[i]
         for (int i = 0; i < n; i++)
            tat[i] = bt[i] + wt[i];
      }
      // Function to calculate average waiting and turn-around
      // times.
      static void findavgTime(int[] processes, int n, int[] bt, int[] at)
      {
         int[] wt = new int[n]; int[] tat = new int[n];

         // Function to find waiting time of all processes
```

```
        findWaitingTime(processes, n, bt, wt, at);

        // Function to find turn around time for all processes
        findTurnAroundTime(processes, n, bt, wt, tat);

        // Display processes along with all details
        Console.Write("Processes " + " Burst Time " + " Arrival Time "
           + " Waiting Time " + " Turn-Around Time "
           + " Completion Time \n");
        int total_wt = 0, total_tat = 0;
        for (int i = 0; i < n; i++)
        {
           total_wt = total_wt + wt[i];
           total_tat = total_tat + tat[i];
           int compl_time = tat[i] + at[i];
           Console.WriteLine(i + 1 + "\t\t" + bt[i] + "\t\t"
              + at[i] + "\t\t" + wt[i] + "\t\t "
              + tat[i] + "\t\t " + compl_time);
        }

        Console.Write("Average waiting time = "
           + (float)total_wt / (float)n);
        Console.Write("\nAverage turn around time = "
           + (float)total_tat / (float)n);
     }
     // Driver code
     public static void Main(String[] args)
     {
        // Process id's
        int[] processes = { 1, 2, 3 };
        int n = processes.Length;
        // Burst time of all processes
        int[] burst_time = { 5, 9, 6 };
        // Arrival time of all processes
        int[] arrival_time = { 0, 3, 6 };
        findavgTime(processes, n, burst_time, arrival_time);
        Console.ReadLine();
     }
  }
}
```

Output:

Processes	Burst Time	Arrival Time	Waiting Time	Turn-Around Time	Completion Time
1	5	0	0	5	5
2	9	3	2	11	14
3	6	6	8	14	20

Average waiting time = 3.33333

Average turn around time = 10.0

II. Điều phối CPU theo thuật toán Round Robin

How to implement in programming language

1. Create two arrays of burst time res_b[] and of
   arrival time res_a[] and copy the value of
   the b[] and a[] array for calculate the
   remaining time.(b[] is burst time, a[] arrival time).
2. Create an another array for wt[] to store waiting time.
3. Initialize Time : t=0;
4. Keep traversing the all process while all
   process are not done.
   Do following for i'th process if it is not done yet.
   a- if res_a[i]<= q (quantum time :- q)
      1. if res_b[i]>q
         a. t=t+q
         b. res_b[i]-=q;
         c. a[i]+=q;
      2. else res_b[i]<=q(for last to execute)
         a. t=t+b[i];
         b. wt[i]=t-b[i]-a[i];
         c.res_b[i]=0;
   b- else res_a[i]>q
      1. Initialize j=0 to number of process
         if a[j]<a[i] (compare is there any
         other process come before these process)
            1. if res_b[j]>q
               a. t=t+q

b. res_b[j]-=q;

c. a[j]+=q;

2. else res_b[j]<=q

a. t=t+b[j];

b. wt[j]=t-b[j]-a[j];

c.res_b[j]=0;

2. now we executing the i'th process

1. if res_b[i]>q

a. t=t+q

b. res_b[i]-=q;

c. a[i]+=q;

2. else res_b[i]<=q

a. t=t+b[i];

b. wt[i]=t-b[i]-a[i];

c.res_b[i]=0;

Code C#

```
namespace RoundRobin
{
   class RoundRobin
   {
      public static void roundRobin(String[] p, int[] a,
                                    int[] b, int q)
      {
         // result of average times
         int res = 0;
         int resc = 0;

         // for sequence storage
         String seq = "";

         // copy the burst array and arrival array
         // for not effecting the actual array
         int[] res_b = new int[b.Length];
         int[] res_a = new int[a.Length];

         for (int i = 0; i < res_b.Length; i++)
         {
            res_b[i] = b[i];
            res_a[i] = a[i];
         }

         // critical time of system
         int t = 0;

         // for store the waiting time
         int[] w = new int[p.Length];

         // for store the Completion time
         int[] comp = new int[p.Length];

         while (true)
         {
            Boolean flag = true;
            for (int i = 0; i < p.Length; i++)
            {
```

```
// these condition for if
// arrival is not on zero

// check that if there come before qtime
if (res_a[i] <= t)
{
   if (res_a[i] <= q)
   {
      if (res_b[i] > 0)
      {
         flag = false;
         if (res_b[i] > q)
         {

            // make decrease the b time
            t = t + q;
            res_b[i] = res_b[i] - q;
            res_a[i] = res_a[i] + q;
            seq += "->" + p[i];
         }
         else
         {

            // for last time
            t = t + res_b[i];

            // store comp time
            comp[i] = t - a[i];

            // store wait time
            w[i] = t - b[i] - a[i];
            res_b[i] = 0;

            // add sequence
            seq += "->" + p[i];
         }
      }
   }
   else if (res_a[i] > q)
   {

      // is any have less arrival time
      // the coming process then execute them
```

```
for (int j = 0; j < p.Length; j++)
{

    // compare
    if (res_a[j] < res_a[i])
    {
        if (res_b[j] > 0)
        {
            flag = false;
            if (res_b[j] > q)
            {
                t = t + q;
                res_b[j] = res_b[j] - q;
                res_a[j] = res_a[j] + q;
                seq += "->" + p[j];
            }
            else
            {
                t = t + res_b[j];
                comp[j] = t - a[j];
                w[j] = t - b[j] - a[j];
                res_b[j] = 0;
                seq += "->" + p[j];
            }
        }
    }
}

// now the previous porcess according to
// ith is process
if (res_b[i] > 0)
{
    flag = false;

    // Check for greaters
    if (res_b[i] > q)
    {
        t = t + q;
        res_b[i] = res_b[i] - q;
        res_a[i] = res_a[i] + q;
        seq += "->" + p[i];
    }
    else
```

```
                {
                    t = t + res_b[i];
                    comp[i] = t - a[i];
                    w[i] = t - b[i] - a[i];
                    res_b[i] = 0;
                    seq += "->" + p[i];
                }
            }
        }
    }

    // if no process is come on thse critical
    else if (res_a[i] > t)
    {
        t++;
        i--;
    }
  }

  // for exit the while loop
  if (flag)
  {
    break;
  }
 }

 Console.WriteLine("name   ctime   wtime");
 for (int i = 0; i < p.Length; i++)
 {
    Console.WriteLine(" " + p[i] + "\t" +
                      comp[i] + "\t" + w[i]);

    res = res + w[i];
    resc = resc + comp[i];
 }

 Console.WriteLine("Average waiting time is " +
                   (float)res / p.Length);
 Console.WriteLine("Average compilation time is " +
                    (float)resc / p.Length);
 Console.WriteLine("Sequence is like that " + seq);
}
```

```
    // Driver Code
    public static void Main(String[] args)
    {
        // name of the process
        String[] name = { "p1", "p2", "p3", "p4" };

        // arrival for every process
        int[] arrivaltime = { 0, 1, 2, 3 };

        // burst time for every process
        int[] bursttime = { 10, 4, 5, 3 };

        // quantum time of each process
        int q = 3;

        // cal the function for output
        roundRobin(name, arrivaltime, bursttime, q);
        Console.ReadLine();
    }
  }
}
```

Output:

```
name  ctime  wtime
 p1    22    12
 p2    15    11
 p3    16    11
 p4    9    6
Average waiting time is 10.0
Average compilation  time is 15.5
Sequence is like that ->p1->p2->p3->p4->p1->p2->p3->p1->p1
```

III. Yêu cầu về nhà: Viết chương trình điều phối theo độ ưu tiên độc quyền và Short Job First độc quyền.