Non-Preemptive Scheduling

Saikot Das Joy
Dept. of Electronic Engineering
Hochschule Hamm-Lippstadt
Hamm,Germany
saikot-das.joy@stud.hshl.de

***Abstract*—This is a seminar work about Non-Preemptive Scheduling where once a process is allocated to resources, the process doesn't leave it until it completes its task or switches to the waiting state It is a simple method and it tends to offer high throughput. In this scheduling system, too many resources are not necessary for scheduling. One of the examples of non-Preemptive scheduling is 'Timer'. This paper explains and elaborates on the importance and methodology, principles, Pros and Cons, and mathematical overview of Non-Preemptive scheduling. How Non-Preemptive Scheduling simplifies the access to shared resources is explained in this paper.**

I. Introduction

Scheduling strategy is one of the important elements of resource management. In example, For a uniprocessor, it is important both for the manufacturers as well as for the owner of the component. It is not only complex to develop a resource for fast handling multiple tasks together but also it costs more. So, Resource management helps to come as a solution for this. this resource management for multiple tasks is named as "Scheduling". There are two types of Scheduling techniques. One is Pre-emptive where one running tasks can be paused to run another more prioritized tasks or process. Another is Non pre-emptive scheduling . In non-Pre-emptive scheduling , once a process is started to execute, another process can never start running until the previous one finishes executing. It is globally accepted that Pre-emptive scheduling is better for a uniprocessor. But, In some cases, Non Pre-emptive scheduling is mandatory such as Timer. This type of scheduling is useful in some real time scenarios. It can be either handling periodic tasks or non periodic tasks. In this seminar work, It is going to be discussed how non pre-emptive scheduling works on a uniprocessor for non periodic tasks. It is also been simulated using C code .

II. Different types of non-preemptive scheduling Techniques

In this Part, Different types of scheduling techniques for non-preemptive scheduling will be discussed.
Formulas:
Completion time = time at which process finishes executing
Turnaround time = completion time-arrival time
Waiting time =turnurnaround time-burst time
Average turnaround time = sum of the turnaround time of all processes/Total Processes
For the approach, Let's assume that the following informations are given:
(1)Arrival time.
(2) Burst time.
where,
Arrival Time = Time at which process arrives.
Burst time = necessary time required to execute the process .

We target to find the followings:
(1)Completion Time.
(2)Waiting Time
(3)Turnaround Time.
(4)Average Turnaround Time

A. *First Come First Serve(FCFS)*

This type of non-pre-emptive scheduling depends on the arrival time of the process completely. The process which arrives earlier, is scheduled to finish earlier without interruption.
Lets consider the following scenario:
in figure 1, you can see, there is 4 processes P1,P2,P3 and

Processes	Arrival time	Burst time
P1	0	3
P2	4	2
P3	2	4
P4	5	1

Figure 1. Processes to be schedule with FCFS

P4 are given with their respective Arrival time and Burst time.Our target is to schedule this processes on a First Come First Serve basis.

So, scheduling will be as below according to the lowest arrival time:

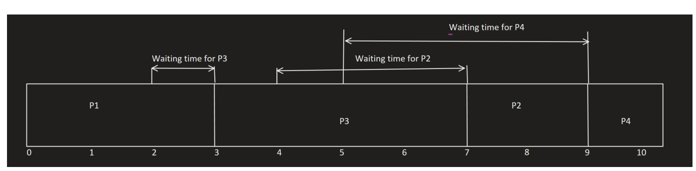

Figure 2. Scheduling diagram for FCFS

in figure 2, which processes is scheduled after which process are shown with their waiting time indicated with a time label.

Processes	Arrival time	Burst time	Completion time	Turnaround time	Waiting time
P1	0	3	3	3	0
P3	2	4	7	5	1
P2	4	2	9	5	3
P4	5	1	10	5	4

Figure 3. Scheduling data for FCFS of Figure 1

in figure 3, completion time, turnaround time and waiting time is calculated for processes P1,P2,P3 and P4.
So average turnaround time is : (3+5+5+5)/4=4.5
Average waiting time is : (0+1+3+4)/4=2

Pros – relative importance of each processes may be defined through this schedulic technique

Cons – Deciding which priority level is assigned to which process is hard. Besides, if a process with high priority with high burst time keeps running, lower priority process may continue to starving for a indefinite time.

B. *Shortest Job First(SJF)*

This scheduling technique doesn't depend on arrival time, but instead it depends on the burst time of the process. However, the process which arrives at the beginning where no other process still didn't arrive will be processed first. Let's consider the following scenario:

in Figure 4, 4 processes P1,P2,P3 and P4 are considered

Processes	Arrival time	Burst time
P1	0	4
P2	5	1
P3	2	3
P4	3	2

Figure 4. Processes to be Scheduled with SJF

with their respective arrival and burst time. So according to the Shortest job algorithom, this 4 processes are scheduled as shown in Figure 5 where waiting time for processes are also noticible as well as time label is given for better understanding. So Scheduling approach with all data are

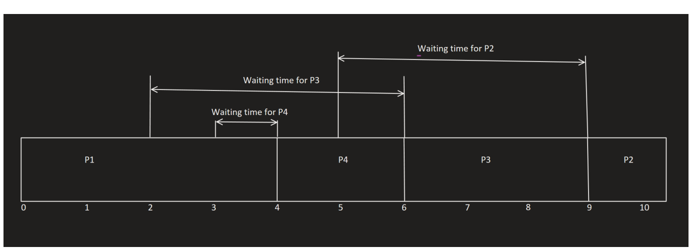

Figure 5. Scheduling diagram for SJF of Figure 4

calculated below:

Processes	Arrival time	Burst time	Completion time	Turnaround time	Waiting time
P1	0	4	4	4	0
P4	3	2	6	3	1
P3	2	3	9	7	4
P2	5	1	10	5	4

Figure 6. Scheduling data for SJF of Figure 4

in Figure 6, respective completion time turnaround time and waiting time is calculated for each processes.

So average turnaround time is : (4+3+7+5)/4=4.75
Average waiting time is : (0+1+4+4)/4=2.25

Pros :
Shortest job gets more priority. For a set of given processes, it gives the minimum average waiting time.

Cons:
If shorter process keeps coming, processes with more burst time will keep starving.

C. *Largest Job First(LJF)*

It is quite similar with the Shortest job First algorithm(SJF) the only difference is, If more processes are on waiting state, instead of the process with minimum burst time, the process with maximum burst time gets the priority and goes to the execution state earlier.

Pros :
Longest job gets more priority. For a set of given processes, it gives the maximum average waiting time.

Cons:
If longer process keeps coming, processes with more burst time will keep starving.

D. *Highest Priority*

In this type of non preemptive scheduling, Priority number is given. Highest priority number can be considered to schedule next process or lower priority number can be considered to schedule next process. Here, We will consider the process with priority number 0 to give maximum priority to a process.
Lets consider the following scenario in FIgure 7.

in Figure 7, four processes P1,P2,P3 and P4 are considered with their arrival time, burst time and priority number , and these processes have to be scheduled based on Highest priority.

in Figure 8, they are scheduled based on their priority number , and waiting time for individual processes are shown with a time label for better understanding. Figure 9 shows the data of calculating the completion time, turnaround time and waiting time of each processes individually.

Processes	Arrival time	Burst time Or Service time	Priority number
P1	0	5	2
P2	3	3	4
P3	5	4	0
P4	6	2	3

Figure 7. Processes to be scheduled with Highest priority number

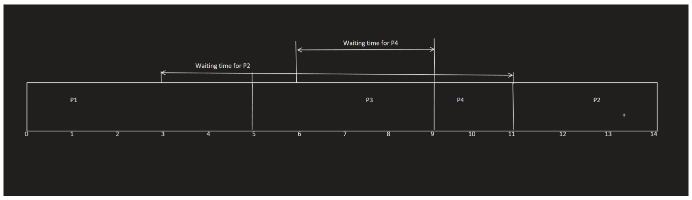

Figure 8. Scheduling diagram for Scheduling with Highest Priority number of Figure 7

So, average waiting time is : (0+0+3+8)/4=2.75
Average Turnaround time =(5+4+7+9)/4=6.25

Pros:
Relative importance of each processes may be defined through this scheduling technique.

Cons:
Deciding which priority level is assigned to which process is hard. Besides, If a process with high priority with high burst time keeps running, lower priority process may continue to starving for a indefinite time.

E. Hghest response ratio next (HRRN)

This type of scheduling doesn't depend only on the burst time(or service time) , But also depends on the waiting time.
Response ratio =(waiting time+service time)/service time
is calculated. This waiting time is calculated at the point of scheduling not like the waiting time discussed in other scheduling techniques before.

Lets consider the following scenario in Figure 10.

Figure 10 shows data of 4 processes which need to be scheduled with HRRN. There is burst time and arrival time given for 4 processes P1,P2,P3 and P4.

So, Since P1 arrives at time 0, It will be executed first. But When P1 finishes executing , Which will be the next process? There are two process P2 and P3 which arrived within this time. At this point,
Waiting time for P2 is 3 and waiting time for P3 is 1.

So,
Response ratio for P2 is = (3+3)/3=2

Processes	Arrival time	Burst time	Completion time	Turnaround time	Waiting time
P1	0	3	5	5	0
P3	5	4	9	4	0
P4	4	6	11	7	3
P2	5	3	14	9	8

Figure 9. Scheduling data for Processes shown in Figure 7

Processes	Arrival time	Burst time Or Service time
P1	0	6
P2	3	3
P3	5	4
P4	6	2

Figure 10. Processes to be scheduled with HRRN

Response time for P3 is = (1+4)/4=1.25

Since, response ratio of P2 is higher than That of P3, P2 will get higher priority and will be scheduled next.
when execution of P2 is complete, the time is 9, within this mean time another Process P4 arrived.
And till this moment , waiting time for P3 is 4. And waiting time for P4 is 3.

Therefore, Response ratio of P3 is= (4+4)/4=2.
And response ratio for P4 is =(3+2)/2= 2.5

Since, response ratio for P4 is higher than P3, P4 will be scheduled next, and P3 will be scheduled after that.

So, the scheduling is shown in Figure 11

Figure 11 explains how each processes are scheduled

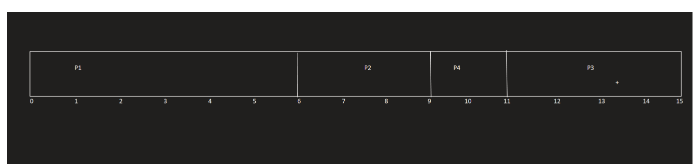

Figure 11. Scheduling diagram for HRRN of Figure 10

one after another with HRRN approach.

III. Implementation of Highest Response Ratio Next (HRRN) Scheduling in C

To have a look on complete code, Please find it on Appendix.
A portion of code which can be interesting is discussed below:

```
// Define the details of the processess
```

```
struct Job {
 char name;
 int Arr_Time, Burst_Time,  Waiting_Time, Turn_Time;
 int Com_Status;
} a[10];
```

you can see that, a structure Array of size 10 is declared. We go for this kind of variables to minimize and make the code more simple as we have no idea how many processes may arrive.

Now, Lets consider the following part of the code given below:

```
   // Check if the process has arrived and is Incomplete
   if (a[i].Arr_Time <= t && a[i].Com_Status!= 1) {

    // Calculating the Response Ratio
    rr = (a[i].Burst_Time + (t - a[i].Arr_Time)) / a[i].Burst_Time;

    // Checking for the Highest Response Ratio
    if (hrr < rr) {

     // Storing the Response Ratio
     hrr = rr;

     // Storing the  Location
     location = i;
    }
   }
```

Here, completion status of process is considered for scheduling. when a process is completed executing, the value of variable com_status turns 1. if it is not 1, then this process has to be scheduled which you can see in the code where com_status=1 or com_status=0 is checked. if com_status=0 , response ratio is calculated for it.

IV. ADVANTAGES OF NON-PREEMPTIVE SCHEDULING

- Low Scheduling overload.
- Conceptually simple.
- throughput is high.
- Less computational resources neccessary.

V. DIS-ADVANTAGES OF NON-PREEMPTIVE SCHEDULING

- Starvation can happen for real time scenarios.
- Process response time can be very poor.
- If a bug comes, it can freeze the whole system.
- Realtime, priority scheduling can be difficult with it.

VI. SUMMERY

There are several type of scheduling techniques among them all techniques are not perfect for all purposes. So for Non-Preemptive scheduling .Though there are some restrictions of this scheduling techniques like scheduling in uniprocessor for short but hard deadline processes, this scheduling approach is also perfect for some hard real time cases where process interruption can result in catastrophic ending. this paper is just a overview of this scheduling approach which can be extended and applied for more and more specific conditions .

REFERENCES

[1] Jian-Jia Chen& Georg von der Brüggen, Non-Preemptive and Limited PreemptiveScheduling. TU Dortmund(2017)

[2] Jayachandran,Praveen,Abdelzaher&Tarek F.(2009). The Case for Non-Preemptive Scheduling in Distributed Real-Time Systems. Find here: http://hdl.handle.net/2142/11332

[3] Schwiegeishohn, U., Yahyapour, R. (1998). Improving first-come-first-serve job scheduling by gang scheduling. Find here: https://doi.org/10.1007/BFb0053987

[4] Latip, R., & Idris, Z. (2011). Highest Response Ratio Next (HRRN) vs First Come First Served (FCFS) Scheduling Algorithm in Grid Environment. ICSECS.

[5] Ru, Jia and Jacky Wai Keung. "An Empirical Investigation on the Simulation of Priority and Shortest-Job-First Scheduling for Cloud-Based Software Systems." 2013 22nd Australian Software Engineering Conference (2013): 78-87.

[6] Putra, Tri. (2020). Analysis of Preemptive Shortest Job First (SJF) Algorithm in CPU Scheduling. IJARCCE. 9. 41-45. 10.17148/IJARCCE.2020.9408.

[7] Yoo, Seong-Moo and Hee Yong Youn. "Largest-job-first-scan-all scheduling policy for 2D mesh-connected systems." Proceedings of 6th Symposium on the Frontiers of Massively Parallel Computation (Frontiers '96) (1996): 118-125.

[8] PusUP. Singh, V. Singh, and A. Pandey, Analysis and Comparison of CPU Scheduling Algorithms. Available from: https://www.ukessays.com/essays/computer-science/a-comparison-of-cpu-scheduling.php?vref=1 [Accessed 2 May 2022].

[9] Raheja, Supriya & Dadhich, Reena & Rajpal, Smita. (2014). Designing of 2-Stage CPU Scheduler Using Vague Logic. Advances in Fuzzy Systems. 2014. 1-10. 10.1155/2014/841976.

[10] Shoaib, Mohammad & Farooqui, Mohd. (2014). A Comparative Review of CPU Scheduling Algorithms. 10.13140/RG.2.1.1748.3680.

[11] https://www.guru99.com/preemptive-vs-non-preemptive-scheduling.html

VII. APPENDIX

Complete impementation code(in C) for HRRN shceduling

```

#include <stdio.h>

// Define the details of the processess
struct Job {
 char name;
 int Arr_Time, Burst_Time,  Waiting_Time, Turn_Time;
 int Com_Status;
} a[10];

int q;

//Sort processes with arrival time (which come earlier?)
void Sorting()
{
 struct Job tempo;
 int i, j;

 // apply selection
 for (i = 0; i < q - 1; i++) {
  for (j = i + 1; j < q; j++) {

   // check which process  arrives earlier
   if (a[i].Arr_Time > a[j].Arr_Time) {

    //swapping
    tempo = a[i];
    a[i] = a[j];
    a[j] = tempo;
   }
  }
 }
}

```

```c
void main()
{
 int i, j, t, Sum_Bt = 0;
 char c;
 char e;
 char d[q];
 float Avg_Waiting_Time = 0, Avg_TurnAr_Time = 0;


 // take input of the number of processes to be scheduled
 printf("Enter number of Processes to be scheduled : ");
 scanf("%i",&q);

 int Arrival[q] ;
 int burst[q] ;

 // taking input for processes
 for(i=0,e='A';i<q;i++,e++){

     d[i]=e;
     printf("Enter Arrival time of Process %c",d[i]);
     printf(": ");
        scanf("%d",&Arrival[i]);

        printf("Enter Burst Time of Process %c",d[i]);
        printf(": ");
        scanf("%d",&burst[i]);
                      }



 // Initializing structure variables
 for (i = 0, c = 'A'; i < q; i++, c++) {
  a[i].name = c;
  a[i].Arr_Time = Arrival[i];
  a[i].Burst_Time = burst[i];

  // Variable for Completion status
  // for Pending = 0
  // for Completed = 1
  a[i].Com_Status = 0;

  // the Variable for the sum of all Burst Times
  Sum_Bt += a[i].Burst_Time;
 }

 // Let us Sort the structure by the arrival times
 Sorting();
 printf("\nName\tArrival Time\tBurst Time\tWaiting Time");
 printf("\tTurnAround Time");
 for (t = a[0].Arr_Time; t < Sum_Bt;) {

  // Now Set the lower limit to response ratio
  float hrr = -9999;

  //The Response Ratio Variable
  float rr;

  // Variable used to store the next processs selected
  int location;
  for (i = 0; i < q; i++) {

   // Check if the process has arrived and is Incomplete
   if (a[i].Arr_Time <= t && a[i].Com_Status!= 1) {

    // Calculating the Response Ratio
    rr = (a[i].Burst_Time + (t - a[i].Arr_Time)) / a[i].Burst_Time;

    // Checking for the Highest Response Ratio
    if (hrr < rr) {

     // Storing the Response Ratio
     hrr = rr;

     // Storing the  Location
     location = i;
    }
   }
  }

  // Updating  time value
  t += a[location].Burst_Time;

  // waiting time
  a[location].Waiting_Time = t - a[location].Arr_Time - a[location].Burst_Time;


  // Turn Around Time
  a[location].Turn_Time = t - a[location].Arr_Time;

  // Sum of Turn Around Time for the average
  Avg_TurnAr_Time += a[location].Turn_Time/q;





  // Updating  the Completion Status
  a[location].Com_Status = 1;

  // Sum of te Waiting Time to calculate the average
  Avg_Waiting_Time += a[location].Waiting_Time/q;
  printf("\n%c\t\t%d\t\t", a[location].name, a[location].Arr_Time);
  printf("%d\t\t%d\t\t", a[location].Burst_Time, a[location].Waiting_Time);
  printf("%d\t\t", a[location].Turn_Time);
 }
 printf("\nverage waiting time:%f\n", Avg_Waiting_Time);
 printf("Average Turn Around time:%f\n", Avg_TurnAr_Time);
}
```

Saikot Das Joy :

Eidesstattliche Erklärung

Hiermit bestätige ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen sowie Hilfsmittel genutzt habe. Alle Ausführungen, die anderen Quellen im Wortlaut oder dem Sinn nach entnommen wurden, sind deutlich kenntlich gemacht. Außerdem versichere ich, dass die vorliegende Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Affidavit

I hereby confirm that I have written this paper independently and have not used any sources or aids other than those indicated. All statements taken from other sources in wording or sense are clearly marked. Furthermore, I assure that this paper has not been part of a course or examination in the same or a similar version.

Joy,Saikot Das	Hamm,11/05/2022	Saikot Das Joy
Name,Vorname	Ort,Datum	Unterschrift
Last name,First name	Location,Date	Signature