**What’s non-preemptive and what’s non-preemptive priority?**

**• Non-preemptive:** Non-preemptive Scheduling is employed when a process has switched from running to waiting state, or when a process has been terminated. During this scheduling, once the resources (CPU cycles) is allocated to a process, the method holds the CPU till it gets terminated or it reaches a waiting state. Just in case of non-preemptive scheduling doesn't interrupt a process running CPU in middle of the execution. Instead, it waits till the method complete its CPU burst time so it can allocate the CPU to a different process. Algorithms supported non-preemptive scheduling are: Shortest Job First (SJF basically non preemptive) and Priority (non -preemptive version), etc.

• **Non-preemptive priority**: In the Non Preemptive Priority scheduling, The Processes are scheduled consistent with the priority number assigned to them. Once the process gets scheduled, it'll run till the completion. Generally, the lower the priority number, the upper is that the priority of the process. The people might get confused with the priority numbers, hence within the GATE, there clearly mention which one is that the highest priority and which one is that the lowest one.

**The advantages and the disadvantages**

**Advantages:** This provides a good mechanism where the relative importance of each process maybe precisely defined.

**Disadvantages:** If high priority processes assign plenty of CPU time, lower priority processes may starve and be postponed indefinitely. The case where a process never gets scheduled to run is named starvation. Another problem is deciding which process gets which priority level assigned to that.

**We will discuss our algorithm**

First of all, we used ASP.net using C# programming language. Also, we used MVC because of many reasons; faster development process, Ability to provide multiple views, support for asynchronous technique, the modification does not affect the entire model, MVC model returns the data without formatting, and SEO friendly Development platform.

**The general idea:** Firstly, we take from the user the number of processes. Secondly, we take the value for each process. Thirdly, we put those data in an array and we arrange this array according to the their arrival time and priority. Last but not least, we calculate the waiting time, turnaround time, response time for each process after we finish that we will get the average wait time, average turnaround time, and average response time for all of the processes.

**We have 2 models:**

* First model which is Process has 4 variables

public List<ProData> ProList { get; set; }

[Required]

[Range(1, int.MaxValue)]

public int NumberOfProcess { get; set; }

public int AverageWaitTime { get; set; }

public int AverageTurnaroundTime { get; set; }

public int AverageResponseTime { get; set; }

I will use these variables in calculating stuff so I need storage in the memory for these variables.

[Required]

[Range(1, int.MaxValue)]

These 2 lines for validation; Required means that the user must fill these variables and the range to prevent the user from inserting negative number.

public List<ProData> ProList { get; set; }

This is a list for each process and this list is another model, which is the second model we have here that contains.

* Second model

[Required]

[Rage(0, int.MaxValue)]

publnic int ProcessId { get; set; }

[Required]

[Range(0, int.MaxValue)]

public int ProcessArrivalTime { get; set; }

[Required]

[Range(1, int.MaxValue)]

public int ProcessBurstTime { get; set; }

[Required]

[Range(0, int.MaxValue)]

public int ProcessPriority { get; set; }

public int ProcessWaitTime { get; set; }

public int ProcessTurnaround { get; set; }

public int ProcessResponse { get; set; }

These variables that I will take it from the user except 3 ProcessWaitTime, ProcessTurnaround, and ProcessResponse; I will calculate it for each process I have.

**The controller and the functions it has.**

public ActionResult Index()

{

Process asd = new Process();

return View(asd);

}

[HttpPost]

public ActionResult Index(Process asd)

{

return RedirectToAction("Insert", asd);

}

First function is Index which is get and post. Get for take the number of process from the user, apparently, I created an object called “asd” from the model process; this object will have the number of the process. Post will send the number to the function insert to let enter the date for each process.

public ActionResult Insert(Process asd)

{

if (asd.NumberOfProcess ==0) {

return RedirectToAction("Index");

}

return View(asd);

}

[HttpPost]

public ActionResult Insert(Process asd, int i = 0)

{

asd = Order(asd);

asd = prodata(asd);

asd.AverageWaitTime=Convert.ToInt32(asd.ProList.Average(s => s.ProcessWaitTime));

asd.AverageTurnaroundTime=Convert.ToInt32(asd.ProList.Average(s => s.ProcessTurnaround));

asd.AverageResponseTime=Convert.ToInt32(asd.ProList.Average(s => s.ProcessResponse));

return View("Result", asd);

}

Insert has get and post. Get will return the view for the user to enter the data for each process; I mean the front end of the project. This code

if (asd.NumberOfProcess ==0) {

return RedirectToAction("Index");

}

For the validation. If the number of process is 0, this means that the user didn’t insert any data so the “if” condition will return the Index again to let him enter the data.

Post has some functions

public ActionResult Insert(Process asd, int i = 0)

{

asd = Order(asd);

asd = prodata(asd);

asd.AverageWaitTime=Convert.ToInt32(asd.ProList.Average(s => s.ProcessWaitTime));

asd.AverageTurnaroundTime=Convert.ToInt32(asd.ProList.Average(s => s.ProcessTurnaround));

asd.AverageResponseTime=Convert.ToInt32(asd.ProList.Average(s => s.ProcessResponse));

return View("Result", asd);

}

**Order function:**

public Process Order(Process asd) {

List<ProData> temp1 = new List<ProData>();

List<ProData> temp = new List<ProData>();

int temp3 = 0;

asd.ProList = asd.ProList.OrderBy(s => s.ProcessArrivalTime).ToList();

for (int i = 0; i < asd.ProList.Count;i++)

{

temp3 = temp3 + asd.ProList[i].ProcessBurstTime;

temp=asd.ProList.Where(c=>c.ProcessArrivalTime<=temp3).OrderBy(r=>r.ProcessPriority).ToList();

}

temp1= asd.ProList.Where(m => m.ProcessArrivalTime == asd.ProList[0].ProcessArrivalTime).OrderBy(r => r.ProcessPriority).ToList();

asd.ProList[0] = temp1[0];

for (int i = 1,s=0 ; s < temp.Count;s++, i++) {

if (temp[s].ProcessId != asd.ProList[0].ProcessId)

{

asd.ProList[i] = temp[s];

}

else { i--; }

}

return asd;

}

We have two temporary lists and one variable

List<ProData> temp1 = new List<ProData>();

List<ProData> temp = new List<ProData>();

int temp3 = 0;

asd.ProList = asd.ProList.OrderBy(s => s.ProcessArrivalTime).ToList();

This will arrange the list according to the arrival time using built-in function called “.OrderBy” Now we arranged our list according to the arrival time.

for (int i = 0; i < asd.ProList.Count;i++)

This is a “for” loop and we have variable “i”. This built-in function .Count will get the number of the elements inside the list we have. What we will do now is we will get the burst time for the first process, for example, this process has burst time 6 and there are 3 another process after this process but with arrival time less than 6 so those processes will start according to the priority after the first process being terminated and so on.

temp3 = temp3 + asd.ProList[i].ProcessBurstTime;

This will get the burst time for all of the processes that have been executed and it to the previous burst time to know the arrival time of the arrived processes or the processes that didn’t arrive yet

the first process and save it in the variable “temp3” we will use it later.

temp=asd.ProList.Where(c=>c.ProcessArrivalTime<=temp3).OrderBy(r=>r.ProcessPriority).ToList();

Here we will save all of the process that has less arrival time than the previous process by using the variable “temp3” and we will do that by using this line of code Where(c=>c.ProcessArrivalTime<=temp3). “Where” is a condition to get all of the process but with arrival time less than the burst time of the previous process as I said above. It will keep doing that until we finish the loop and our list being ordered correctly.

temp1= asd.ProList.Where(m => m.ProcessArrivalTime == asd.ProList[0].ProcessArrivalTime).OrderBy(r => r.ProcessPriority).ToList();

This will get the index zero of the “ProList” list because it has the lowest arrival time so this will check if there are many processes have the same arrival time in index zero so it will gather them and arrange them according to the priority. And save it in the list “temp1”

asd.ProList[0] = temp1[0];

This to make sure that the program will start with the first process that has the lowest arrival time.

for (int i = 1,s=0 ; s < temp.Count;s++, i++) {

if (temp[s].ProcessId != asd.ProList[0].ProcessId)

{

asd.ProList[i] = temp[s];

}

else { i--; }

}

This loop to assign the values from the list “temp” to our original list “ProList”

if (temp[s].ProcessId != asd.ProList[0].ProcessId)

{

asd.ProList[i] = temp[s];

}

else { i--; }

This condition to make sure that the program don’t repeat the process that we already saved it here

asd.ProList[0] = temp1[0];

if the program found it, the counter will be decreased by one and continue the loop normally.

**prodata function:**

public Process prodata(Process asd)

{

int s = 0;

foreach (var ele in asd.ProList)

{

ele.ProcessWaitTime = s - ele.ProcessArrivalTime;

ele.ProcessTurnaround = ele.ProcessWaitTime + ele.ProcessBurstTime;

ele.ProcessResponse = ele.ProcessWaitTime - ele.ProcessArrivalTime;

s = s + ele.ProcessBurstTime;

}

return asd;

}

In this function we will calculate process wait time, process turnaround time, and process response

We defined a variable called “s” and assigned to it zero value to calculate the process wait time. These calculations according to the algorithm of the Non-preemptive priority.

We finished the function inside the “insert” function, I will explain the rest of the function insert. We need to calculate the average wait time, average turnaround time, and average response time.

asd.AverageWaitTime=Convert.ToInt32(asd.ProList.Average(s => s.ProcessWaitTime));

asd.AverageTurnaroundTime=Convert.ToInt32(asd.ProList.Average=>s.ProcessTurnaround));

asd.AverageResponseTime=Convert.ToInt32(asd.ProList.Average(s => s.ProcessResponse));

We used built-in functions to calculate the average for each one.

asd.ProList.Average(s => s.ProcessWaitTime) it will get the index of pro list and get the average of the process wait time.

Convert.ToInt32 this for casting.

We will do the same for the last 2 lines.

Eventually it will return the view of the function result and it will take the object “asd”. By the way, this is the function result

public ActionResult Result(Process asd)

{

return View(asd);

}

As I said it will return the view.

Now the object has all of the data that the user inserted it and the data that I calculated it.

**Question Example :**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| process | Arrival Time | CPU Burst Time | Priority | Turnaround Time | Waiting Time | Response Time |
| P0 | 0 | 10 | 5 | 10 | 0 | 0 |
| P1 | 1 | 6 | 4 | 21 | 15 | 14 |
| P2 | 3 | 2 | 2 | 13 | 8 | 8 |
| P3 | 5 | 4 | 0 | 9 | 0 | 0 |
| Average | | | | 13 | 8 | 6 |

|  |  |  |  |
| --- | --- | --- | --- |
| waiting time  P0 => 0 - 0 = 0 P1 => 16 – 1 = 15 P2 => 11 – 3 = 8 P3 => 5 – 5 = 0 | Turnaround Time =waiting Time + Burst Time  P0 => 0 + 10 = 10 P1 => 15 + 6 = 21 P2 => 8 + 2 = 13 P3 => 5 + 4 = 9 | Response Time =First Time enter CPU-Arrival Time  P0 => 0 - 0 = 0 P1 => 15 – 1 = 14 P2 => 11 – 3 = 8 P3 => 5 – 5 = 0 | Average  for waiting Time  ( 0 + 15 + 11 + 5/4=7.7 => 8 ) for Turnaround Time  ( 10 + 9 + 13 + 21 /4 = 13.2 => 13 ) for Response Time  ( 0 + 0 + 8 + 14 / 4 =5.5 =>6 ) |

Gant chart

P0 P3 P2 P1  
0 10 14 16

**Answer//**

Firstly, enter number of process

A screenshot of a social media post

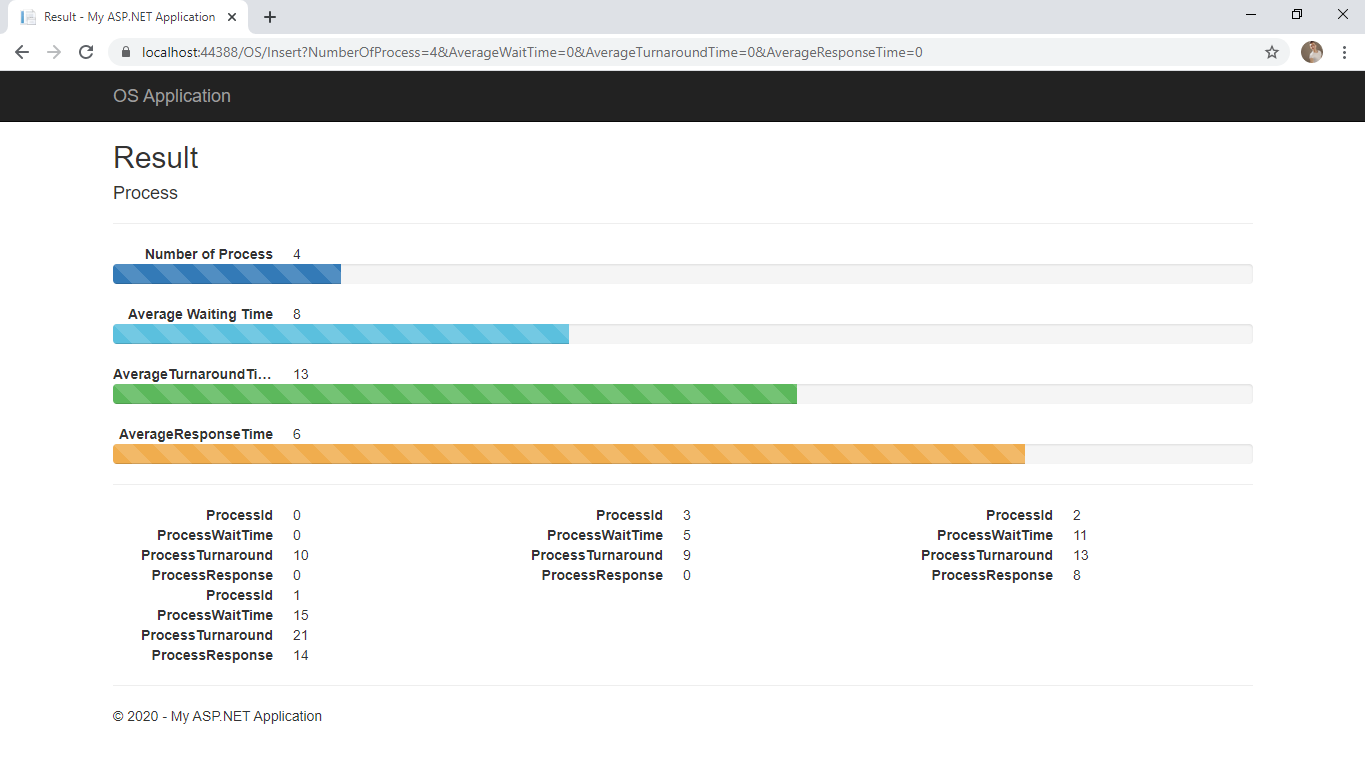
Description automatically generated

Secondly enter process data

A screenshot of a computer screen

Description automatically generated

Finally, the result



**References**

<https://www.javatpoint.com/os-non-preemptive-priority-scheduling>

https://www.geeksforgeeks.org/advantages-and-disadvantages-of-various-cpu-scheduling-algorithms/

<https://www.geeksforgeeks.org/preemptive-and-non-preemptive-scheduling/>

Question:

<http://examradar.com/non-preemptive-priority-scheduling-algorithms-questions-answers/>