**Name: Juhi Patel**

**Preemptive Priority High**

Table of Contents

|  |  |
| --- | --- |
| No | Content |
| 1 | Description of Preemptive priority high (PPH) |
| 2 | Description of Implementation |
| 3 | Experiments |
| 4 | Conclusion |
| 5 | References |

**(1) Description of Preemptive Priority High Algorithm (PPH)**

In preemptive priority high(PPH), higher number represents higher priority. So, at any given time higher priority processes will be executed first. This algorithm is preemptive. That means it can switch between processes. In CPU, processes come at different arrival times. So, in every quantum time, it will check that is there any other higher priority process has arrived? If yes, then CPU will switch to that higher priority process. Otherwise, CPU will continue executing the same process as before. If some process having burst time less than the quantum, then after completing execution of that process CPU will look for other higher priority processes to execute.

**Example:**

|  |  |  |  |
| --- | --- | --- | --- |
| Process | Arrival Time | Burst Time | Priority |
| P1 | 2 | 5 | 3 |
| P2 | 0 | 4 | 2 |
| P3 | 5 | 8 | 5 |
| P4 | 2 | 2 | 1 |

Quantum = 2

Gant Chart:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P2 | P1 | P3 | P1 | P2 | P4 |

0 2 6 14 15 17 19

In the above example, quantum size is 2. At time 0, only P2 is available. So, CPU will start its execution. As quantum time is 2, we will check for new process every 2-time units. So, at time 2, P1 and P4 are arriving. Among P1, P2 and P4, P1 has higher priority. So, CPU will start executing P1. And now P2 has 2 remaining burst time. At time 4, again check for highest priority processes among available processes. So, at time 4, no other new process is coming and P1 has highest priority. So, CPU will continue executing P1. At time 6, P3 is arrived. So, among all 4 processes P3 has highest priority. CPU will start executing P3 and now P1 has 1 remaining burst time. At time 8, no other process arriving and P3 having higher priority. So, it will continue running P3. Same for time 10, 12, 14. At time 14, P3 is done. So, now among available processes p1 has higher priority and it has 1 remaining bust time. At time 15, P1 is done. Now, between P2 and P4, P2 has higher priority. It will be started executing. After P2, P4 will be executed.

**(2) Description of Implementation**

I have implemented this algorithm using two classes.

PPH.java : This class contains main method and other supporting methods startPph and prepareQueue.

PCB.java : This class is to store process information. It has 3 instance variables, arrivalTime, BurstTime and priority and their getter and setter methods.

The pseudocode is as follow:

* Read the process information from file, store them in PCB object and save the reference of PCB in an array.
* Initially, prepare queue of linked list of PCB reference sorted according to the arrival time of processes using prepareQueue method.
* Execute startPph
* In startPph, there is one time variable working as clock, one runningP variable, to track previously running method, one start variable to store execution starting time of a running process.
* while queue is not empty

- create a priority queue of processes which have been arrived at given time and burst time is not 0.

- If this queue is not empty then the first element will have highest priority. So, remove it from priority queue to execute it next. If this removed process is not same as previously running process then that means preemption has occurred. So, write the data for previously running process. Store the start time for new process.

- If the burst time of the process is greater than or equal to quantum then decrement the burst time of the running process by quantum and increment timer by quantum. Else increment the timer by process's burst time and make its burst time 0.

- If burst time become zero then remove it from the queue.

(3) Experiments

Experiment #1:

1. INPUT DATA:



|  |  |  |  |
| --- | --- | --- | --- |
| Process | Arrival Time | Burst Time | Priority |
| P1 | 0 | 8 | 5 |
| P2 | 1 | 4 | 1 |
| P3 | 2 | 9 | 4 |
| P4 | 3 | 5 | 3 |

1. Gantt Chart

|  |  |  |  |
| --- | --- | --- | --- |
| P1 | P3 | P4 | P2 |

0 8 17 22 26

1. Average Waiting Time

= [ P1(0-0) + P2(22-1) + P3(8-2) + P4(17-3) ] / 4 = 10.25

Average Response Time

= [P1(0-0)+P2(22-1)+P3(8-2)+P4(17-3)] / 4 = 10.25

1. SHOULD-BE Output: (e) MY Output (Produced by program)

“output.data” MY “output.data”



Experiment #2:

1. INPUT DATA:



|  |  |  |  |
| --- | --- | --- | --- |
| Process | Arrival Time | Burst Time | Priority |
| P1 | 2 | 5 | 3 |
| P2 | 0 | 4 | 2 |
| P3 | 5 | 8 | 5 |
| P4 | 2 | 2 | 1 |

1. Gantt Chart

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P2 | P1 | P3 | P1 | P2 | P4 |

0 2 6 14 15 17 19

1. Average Waiting Time

= [ P1(2-2)+P1(14-6) + P2(15-2) + P3(6-5) + P4(17-2) ] / 4 = 9.25

Average Response Time

= [P1(2-2) + P2(0-0) + P3(6-5) + P4(17-2)] / 4 = 4.75

1. SHOULD-BE Output: (e) MY Output (Produced by program)

“output.data” MY “output.data”



Experiment #3:

1. INPUT DATA:



|  |  |  |  |
| --- | --- | --- | --- |
| Process | Arrival Time | Burst Time | Priority |
| P1 | 5 | 4 | 5 |
| P2 | 0 | 2 | 6 |
| P3 | 2 | 1 | 7 |
| P4 | 4 | 8 | 4 |

1. Gantt Chart

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P2 | P3 |  | P4 | P1 | P4 |

0 2 3 4 7 11 16

1. Average Waiting Time

= [ P1(7-5) + P2(0-0) + P3(2-2) + P4(11-7) ] / 4 = 1.5

Average Response Time

= [P1(7-5) + P2(0-0) + P3(2-2) +P4(4-4)] / 4 = 0.005

1. SHOULD-BE Output: (e) MY Output (Produced by program)

“output.data” MY “output.data”



**Conclusion:**

Advantage: This algorithm provides a good mechanism where the relative importance of each process may be precisely defined.

Disadvantage : If Higher priority processes keep coming then it can lead to starvation and if the quantum time is very small then it will take more time in checking for available higher priority process and doing switches.

To overcome starvation, aging is the solution and to avoid CPU switching time, quantum time should not be too small and too large.

**References:**

[**http://cs.newpaltz.edu/~phamh/gos/**](http://cs.newpaltz.edu/~phamh/gos/)

[**http://codex.cs.yale.edu/avi/os-book/OS9/slide-dir/index.html**](http://codex.cs.yale.edu/avi/os-book/OS9/slide-dir/index.html)