Danyelle Barrett

OS: Project 1
CPU Scheduling Algorithm
pPH

Table of Contents

pPH (Preemptive Priority High)

1) DESCRIPTION OF SCHEDULING ALGORITHM:

Preemptive Priority High is the scheduling algorithm in which the scheduler ensures that at any given time, the process with the highest priority is executing. (Note: a higher number means higher priority) Since it is preemptive, this means that a process could be taken out during the middle of execution if a process with a higher priority arrives during that time. Also, if a quantum number is specified, the CPU will only check if a new process has a higher priority at the specified quantum increments.

An example of this would be:

Process	Arrival Time	Burst Time	Priority
P1	0	7	2
P2	3	5	1
P3	6	8	8
P4	10	2	5

Gantt Chart:

P1	P3	P4	P4	P2

0 6 14 16 17 22

To explain this example, you can see that P1 gets sent to the CPU first because it is the only process to arrive at time 0. It then only gets executed until time 6, because at time 6, P3 arrives and it has a higher priority. P3 then fully executes because it has the highest priority. Then, the scheduler sends P4 to the CPU because it has the next highest priority. After that, P1 gets sent back to finish executing since it has the next highest priority. Finally, P2 executes to completion since it has the lowest priority.

2) DESCRIPTION OF IMPLEMENTATION:

I went about writing my program in the following way:

- Read input.data line by line, and send each line(string) to create a PCB object. A PCB object has the int attributes of Process Number, Arrival Time, Burst Time, and Priority. All of the PCB objects are placed into an ArrayList called Processes.
- Processes then gets sent to the dispatcher method. The dispatcher simulates how a pPH scheduling method would work. While there are still processes in the ArrayList, the CPU will continue running. Each time through the loop simulates one second of time. Each time around, the Processes ArrayList will be iterated through, and the arrival times will be checked.
- If a process's arrival time matches the current time, then it will be added to a new ArrayList called Arrived.

- From the Arrived ArrayList, the dispatcher will then iterate through the whole list and find the PCB with the highest priority(highest number). This is done by comparing every PCB's priority to a variable called HighestP.
- If at this time (quantum time check) a new PCB has a higher priority, then the HighestP will be switched and set to that PCB which has the highest priority from Arrived.
- After the preemption, the burst time of HighestP is decreased by one to show that it has been executed.
- Then, the information from this cycle is recorded as a printer Object and placed in a print ArrayList.
- Before the loop cycle is finished, the dispatcher has a check to see if the process has finished executing. If it has, then it is removed from both ArrayLists to show that it is done.
- After the while loop has finished and all of the processes have been scheduled and finished executing, then the scheduling data is sent to "output.data". Since every second's data was recorded, we only want to print out the data from when the preemption occurs or when a process finishes. To accomplish that I placed a check when iterating though the print ArrayList to compare the items.
- Finally, the correct schedule data is outputted to "output.data" and the CPU pPH scheduling simulation is complete.
- Note: Additionally, my program has a debugging function that is currently commented out. This line of code outputs to the command line what is occurring at every second of time for debugging purposes.

3) EXPERIMENTS:

Experiment #1:
My Focus: Make sure preemption is working.

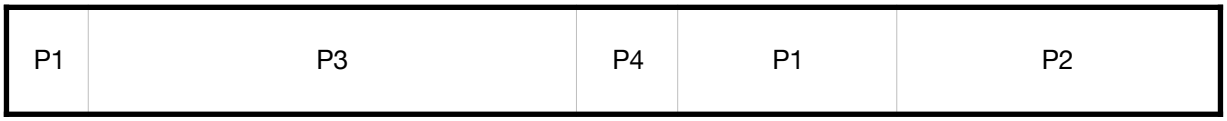
a) INPUT DATA:

"input.data"

```
4

1 1

0 6 2

1 5 1

2 8 8

3 2 5
```

Process	Arrival Time	Burst Time	Priority
P1	0	6	2
P2	1	5	1
P3	2	8	8
P4	3	2	5

b) GANTT CHART (pPH)



0 2 10 12 16 21

c) AVERAGE WAITING TIME
= [P1(12-2) + P2(16-1) + P3(2-2)+ P4(10-3)]/4 = 32/4 = 8

d) SHOULD-BE Output:

```
0 2 1

2 10 3

10 12 4

12 16 1

16 21 2
```

“output.data”

e) MY Output (produced by program):

```
0 2 1

2 10 3

10 12 4

12 16 1

16 21 2
```

My “output.data”

Experiment #2:
My focus: -Scheduler will only do the specified amount of processes.
-With a mixed order of arrival time, it still works properly with preemption.

a) INPUT DATA:

“input.data"

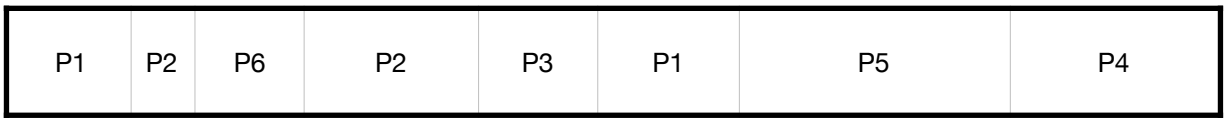
```
6
```

```
1 1
0 5 5
3 4 8
5 2 6
8 3 3
7 4 4
4 2 9
6 6 6
```

Process	Arrival Time	Burst Time	Priority
P1	0	5	5
P2	3	4	8
P3	5	2	6
P4	8	3	3
P5	7	4	4
P6	4	2	9
P7	6	6	6

Note: P7 will never be scheduled because at the top of input.data we specified only 6 processes to be scheduled.

b) GANTT CHART (pPH)



0 3 4 6 9 11 13 17 20

c) AVERAGE WAITING TIME

= [P1(11-3) + P2(3-3)+(6-4) + P3(9-5)+ P4(17-8) + P5(13-7) + P6(4-4)] = 29/6 = 4.83

d) SHOULD-BE Output:

```
0 3 1

3 4 2

4 6 6

6 9 2

9 11 3

11 13 1

13 17 5

17 20 4
```

“output.data”

e) MY Output (produced by program):

My “output.data”

```
0 3 1

3 4 2

4 6 6

6 9 2

9 11 3

11 13 1

13 17 5

17 20 4
```

Experiment #3:

My focus: Make sure it works correctly when Quantum time is not equal to 1.
For this experiment, same input as Experiment #2, but quantum equals 3.

a) INPUT DATA:

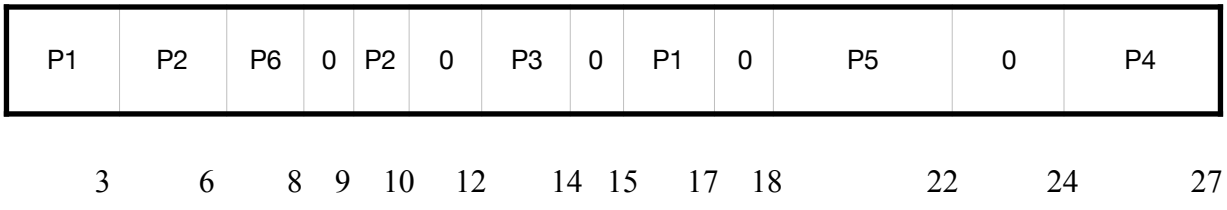
“input.data"

```
6

1 3

0 5 5

3 4 8

5 2 6

8 3 3

7 4 4

4 2 9

6 6 6
```

Process	Arrival Time	Burst Time	Priority
P1	0	5	5
P2	3	4	8
P3	5	2	6
P4	8	3	3
P5	7	4	4
P6	4	2	9
P7	6	6	6

Note: P7 will never be scheduled because at the top of input.data we specified only 6 processes to be scheduled.

b) GANTT CHART (pPH)



Note: ‘0’ means no process is running. This occurs when a process finishes executing before the next quantum time(3) occurs, i.e. when the scheduler is allowed to check for the next process with the highest priority that’s arrived.

c) AVERAGE WAITING TIME

= [P1(0-0)+(15-3) + P2(3-3)+(9-6) + P3(12-5)+ P4(24-8) + P5(18-7) + P6(6-4)]/6 = 51/6 = 8.5

d) SHOULD-BE Output:

“output.data”

```
0 3 1

3 6 2

6 8 6

9 10 2

12 14 3

15 17 1

18 22 5

24 27 4
```

e) MY Output (produced by program):

My “output.data”

```
0 3 1

3 6 2

6 8 6

9 10 2

12 14 3

15 17 1

18 22 5

24 27 4
```

Note: It is easy to tell that the quantum is working correctly, as processes are only being scheduled on multiples of 3, regardless if a process finishes executing before the next quantum time.

4) CONCLUSION:

I found this project to truly test my knowledge not only on programming, but on CPU scheduling algorithms as well. I had to have a complete understanding of the preemptive priority high algorithm before I could even begin this project. I approached coding my program step by step, and through debugging and running many test cases, I eventually got the correct solution to be produced every time! When I did my three experiments, I made sure that they all had different goals, so that I could ensure I was covering all possible cases.

For example, in Experiment #1 I focused on making sure my preemption was running correctly. I did not add any external factors that could influence this.

After I saw that this was working correctly, I then focused Experiment #2 on making sure the scheduler only scheduled the specified amount of processes, even if it was given more. I also listed the processes not in order of their arrival time, so I could make sure my program would still process them in the correct order. Thankfully, it did.

In Experiment #3, I really had to make sure my program could handle anything. This is when I messed around with the quantum time. I kept the same input file from Experiment #2, but I changed the quantum time from 1 to 3. When I saw the output, I could easily tell that it had worked correctly! The start times were all multiples of 3, so I knew that was correct. When I looked at the end times, I could see that some processes finished before the next one started. When I drew the Gantt Chart, this

further proved to be true. There was several holes/gaps where a process had completed, but the next processes couldn't be run until the next quantum time came. I was very happy to see that my pPH simulation program was fully executing correctly, under any given circumstance!

Although this project took some time to complete, I thoroughly enjoyed the challenge. I constantly had to think of possible "what ifs" and make sure they were factored into my coding. In my RAR file, I included everything necessary, and I also submitted the input/output files from experiments 1 and 2. I just had to rename them, so they wont work unless they are changed back to just "input.data" or "output.data". In conclusion, I feel like I have a much greater understanding of how CPU scheduling, preemption, priority scheduling, and quantum times work and I was very happy to complete this project.

5) REFERENCES:

I did not reference any material. I wrote the code from scratch on my own, and I created the experiments on my own. The only material I referenced was the Project Requirements sheet and the Ch. 6 lecture powerpoint slides.