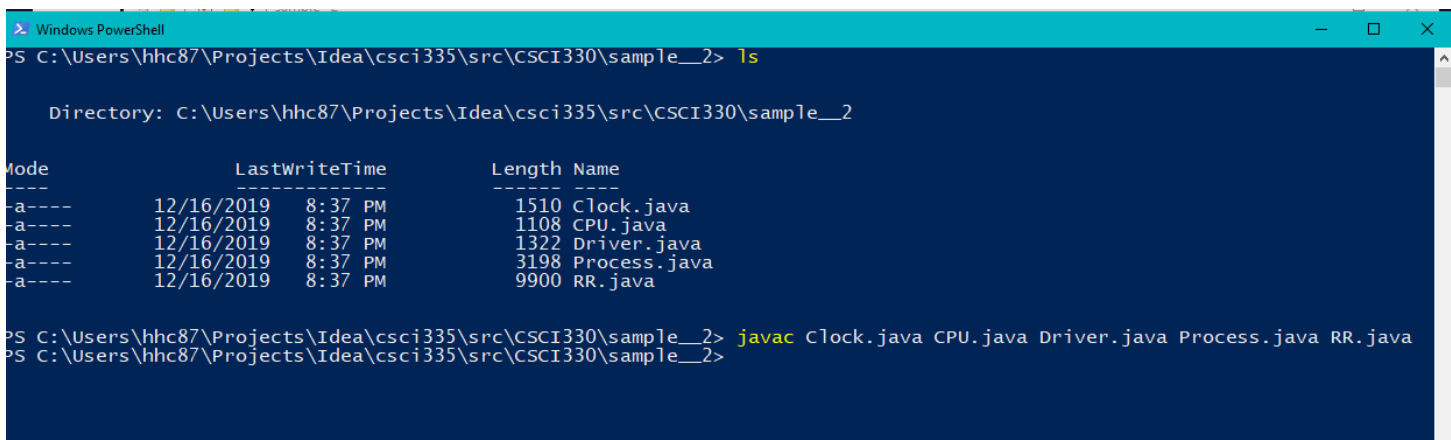


Hui (Henry) Chen
CSCI 330 – M03/ Fall 2019
Dr. Gass
Project – CPU Round Robin Scheduling Algorithm
Dec 17, 2019

Project: CPU Round Robin Scheduling Algorithm

How to run the code?

1. Enter the same directory of the java files and then open the command type
`javac Clock.java CPU.java Driver.java Process.java RR.java`



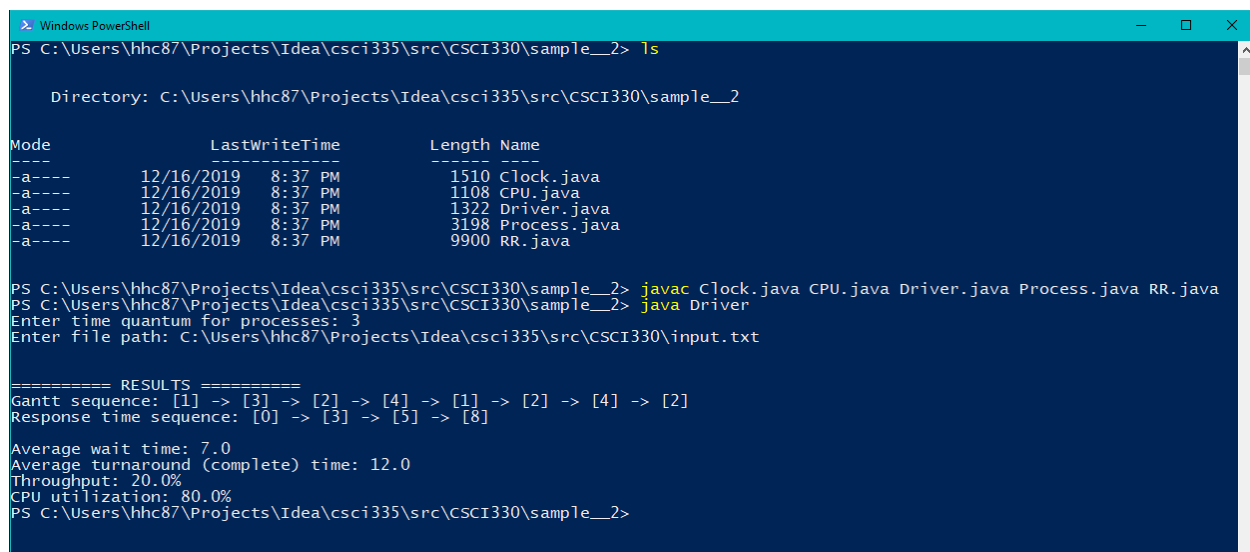
```
Windows PowerShell
PS C:\Users\hmc87\Projects\Idea\csci335\src\CSCI330\sample__2> ls

Directory: C:\Users\hmc87\Projects\Idea\csci335\src\CSCI330\sample__2

Mode                LastWriteTime         Length Name
----                -
-a----          12/16/2019   8:37 PM           1510 Clock.java
-a----          12/16/2019   8:37 PM           1108 CPU.java
-a----          12/16/2019   8:37 PM           1322 Driver.java
-a----          12/16/2019   8:37 PM           3198 Process.java
-a----          12/16/2019   8:37 PM           9900 RR.java

PS C:\Users\hmc87\Projects\Idea\csci335\src\CSCI330\sample__2> javac Clock.java CPU.java Driver.java Process.java RR.java
PS C:\Users\hmc87\Projects\Idea\csci335\src\CSCI330\sample__2>
```

2. Type `java Driver` in order to run the program



```
Windows PowerShell
PS C:\Users\hmc87\Projects\Idea\csci335\src\CSCI330\sample__2> ls

Directory: C:\Users\hmc87\Projects\Idea\csci335\src\CSCI330\sample__2

Mode                LastWriteTime         Length Name
----                -
-a----          12/16/2019   8:37 PM           1510 Clock.java
-a----          12/16/2019   8:37 PM           1108 CPU.java
-a----          12/16/2019   8:37 PM           1322 Driver.java
-a----          12/16/2019   8:37 PM           3198 Process.java
-a----          12/16/2019   8:37 PM           9900 RR.java

PS C:\Users\hmc87\Projects\Idea\csci335\src\CSCI330\sample__2> javac Clock.java CPU.java Driver.java Process.java RR.java
PS C:\Users\hmc87\Projects\Idea\csci335\src\CSCI330\sample__2> java Driver
Enter time quantum for processes: 3
Enter file path: C:\Users\hmc87\Projects\Idea\csci335\src\CSCI330\input.txt

===== RESULTS =====
Gantt sequence: [1] -> [3] -> [2] -> [4] -> [1] -> [2] -> [4] -> [2]
Response time sequence: [0] -> [3] -> [5] -> [8]

Average wait time: 7.0
Average turnaround (complete) time: 12.0
Throughput: 20.0%
CPU utilization: 80.0%
PS C:\Users\hmc87\Projects\Idea\csci335\src\CSCI330\sample__2>
```

Background:

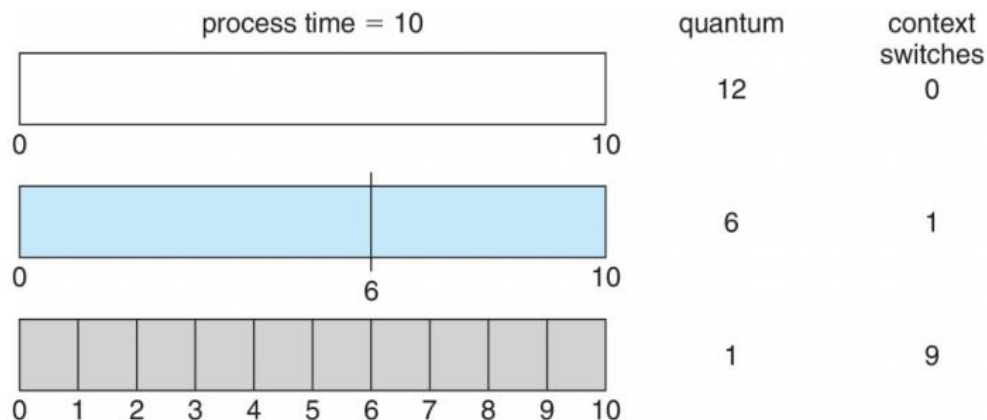
1. CPU Scheduling

CPU scheduling is a process that allows one process at time to use the CPU while the execution of another process is on ready queue due to the unavailability of any resource such as I/O, Direct Memory Access, and etc. CPU scheduling is aimed at making the process effective, fast, and fair.

2. CPU RoundRobin Scheduling

One of CPU scheduling algorithms that uses a variable called time quantum and the impact of the size of time quantum on the overall performance in terms of number of context switch. The time quantum is to get a small unit of CPU time, usually 10 ~ 100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue. Also, this algorithm runs the process based on arrival time in ascending order; therefore, this gives the best response time amongst other scheduling algorithms. However, because the size of time quantum affect the context switch, this creates an overhead problem of context switching (if time quantum is too small).

For example,



this image demonstrates that the lower the time quantum value, the more overhead of context switch.

Implementation:

1. Classes

a. Driver

The user lever and main class of the programm, which gats two inputs from the user: time quantum and file path.

b. Process

The object class which contains all necessary information about a process such as process name, arrival time, burst time, and etc.

c. RR

The impletementation level of the program which has the following methods:

- I. RR – to initialize the things for round robin algorithm
- II. Exeute – to run the RR scheduling algorithm in other methods
- III. Creater_proc – to create a process after preemitting from CPU
- IV. RoundRobin - the core of the programm which contains the algorithm of the RR scheduling
- V. Terminate – to end the scheduling execution which contains most of program calculation
- VI. Chart_reponseT/ Chart_GT – to create a sequence for response time and Gantt chart
- VII. Display_result – return the program result

d. CPU

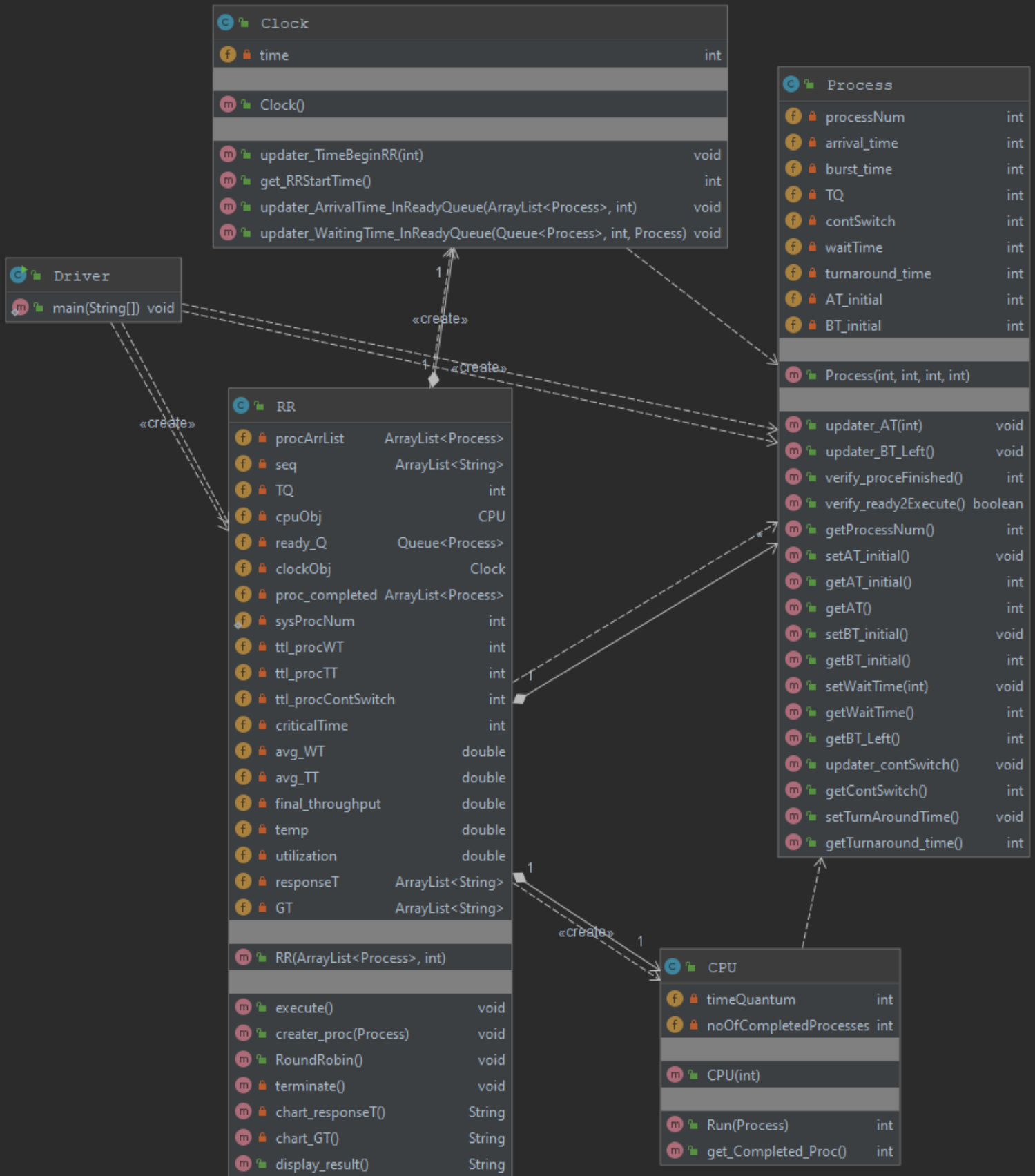
The RR sheduling class helper which helps to determine whether if the current process needs to be executed

e. Clock

The RR scheduling class helper which timing the RR scheduling algorithm for the program

Note: more detail in the code comments

2. UML



Results

1. Input data

1,0,5
2,1,7
3,0,2
4,2,6

2. Screenshots & Explanation

i. Time quantum = 3

```
Driver
Run: Driver (1) x
"C:\Program Files\Java\jdk-11.0.5\bin\java.exe" "-javaagent:C:\Program Files\JetBr
Enter time quantum for processes: 3
Enter file path: C:\Users\hhc87\Projects\Idea\csci335\src\CSCI330\input.txt

===== RESULTS =====
Gantt sequence: [1] -> [3] -> [2] -> [4] -> [1] -> [2] -> [4] -> [2]
Response time sequence: [0] -> [3] -> [5] -> [8]

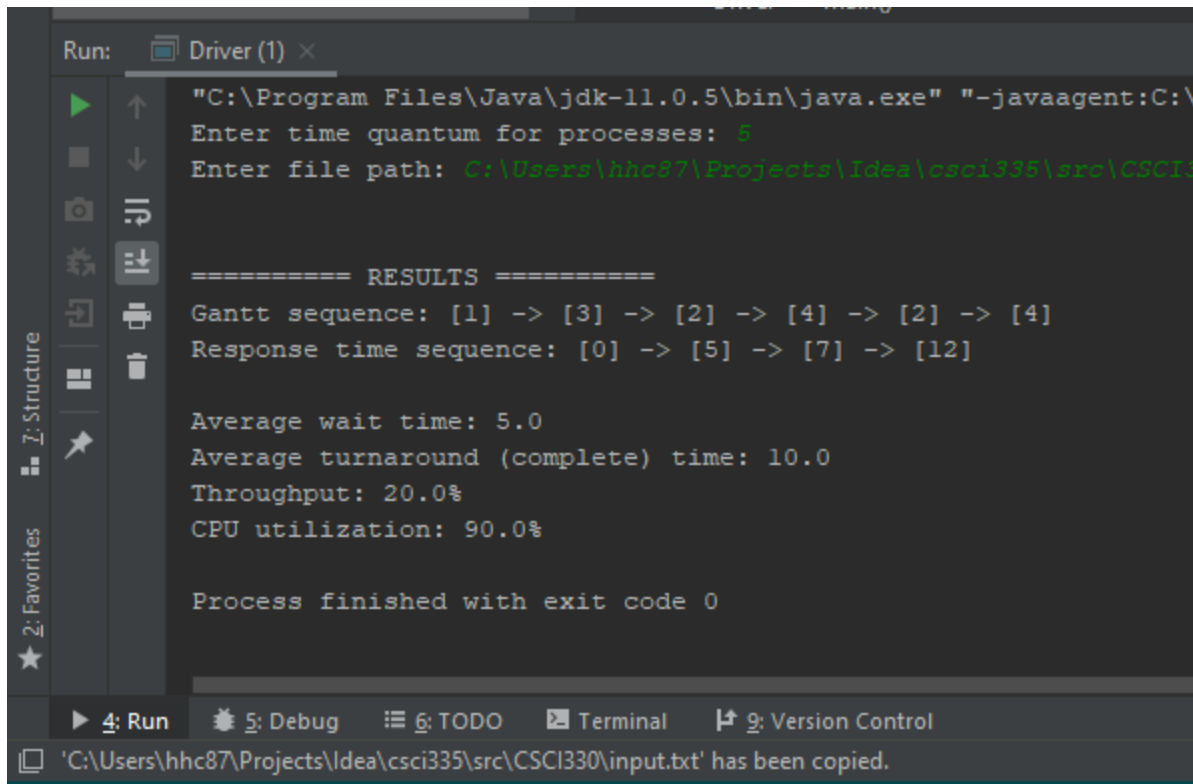
Average wait time: 7.0
Average turnaround (complete) time: 12.0
Throughput: 20.0%
CPU utilization: 80.0%

Process finished with exit code 0

4: Run 5: Debug 6: TODO Terminal Version Control Messages
'C:\Users\hhc87\Projects\Idea\csci335\src\CSCI330\input.txt' has been copied.
```

From the screenshot we can see the average wait time is 7.0, which means a process wait for 7 msec in the ready queue before getting the turn to run in CPU – this is quit long. Also, the average turnaround time is 12.0 msec, which it's quite long, too! The throughput is 20%, and CPU utilization is 80%, which is not so bad because we managed to keep above 75%. However, the time quantum => 3 gives 0, 3, 5, and 8 response time (msec) for P₁, P₃, P₂, and P₄ which is good because that shows how a process response. Also, from Gantt sequence, we can see that total number of context switch is 4, which is acceptable because we have four processes to execute.

ii. Time quantum = 5



```
Run: Driver (1) x
"C:\Program Files\Java\jdk-11.0.5\bin\java.exe" "-javaagent:C:\
Enter time quantum for processes: 5
Enter file path: C:\Users\hmc87\Projects\Idea\csci335\src\CSCI3
===== RESULTS =====
Gantt sequence: [1] -> [3] -> [2] -> [4] -> [2] -> [4]
Response time sequence: [0] -> [5] -> [7] -> [12]

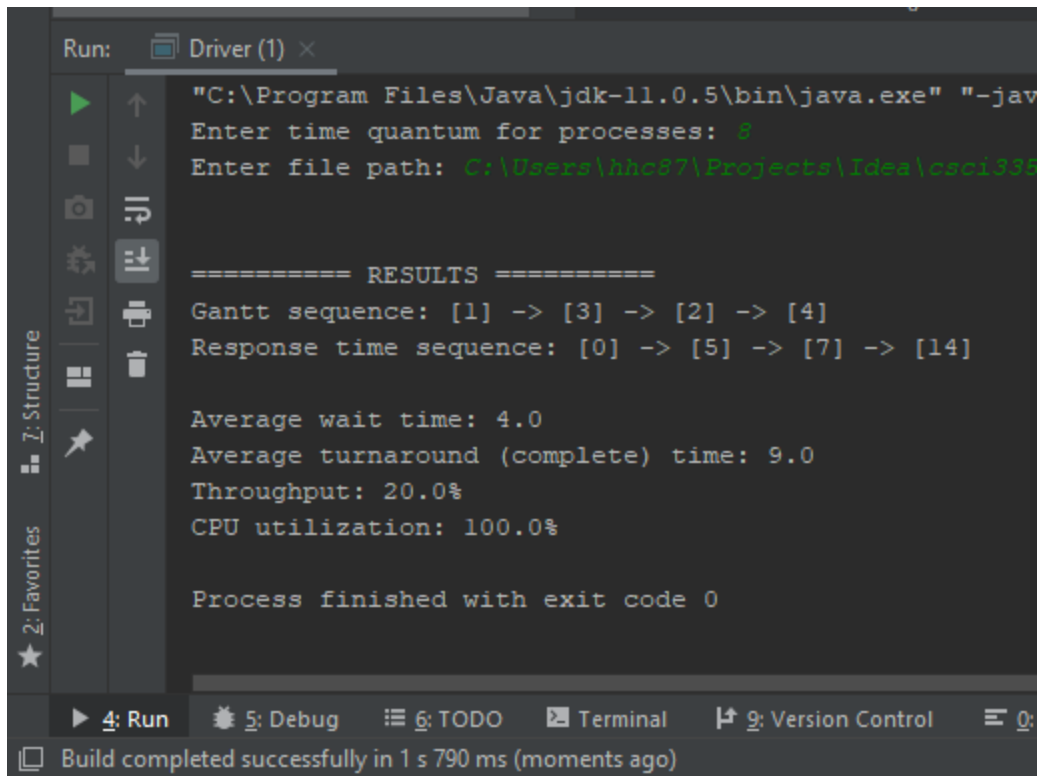
Average wait time: 5.0
Average turnaround (complete) time: 10.0
Throughput: 20.0%
CPU utilization: 90.0%

Process finished with exit code 0

4: Run 5: Debug 6: TODO Terminal 9: Version Control
'C:\Users\hmc87\Projects\Idea\csci335\src\CSCI330\input.txt' has been copied.
```

From the screenshot we can see we get a much better average wait time and turnaround time compared to time quantum 3. Although the throughput is still remaining 20%, the CPU utilization is significantly high because ideally, we try to keep CPU 100% but not in real life. Therefore, this is not very efficient. Also, the response time for each process gets greater compare to time quantum 3.

iii. Time quantum = 8



```
Run: Driver(1) x
"C:\Program Files\Java\jdk-11.0.5\bin\java.exe" "-jav
Enter time quantum for processes: 8
Enter file path: C:\Users\hho87\Projects\Idea\csai338

===== RESULTS =====
Gantt sequence: [1] -> [3] -> [2] -> [4]
Response time sequence: [0] -> [5] -> [7] -> [14]

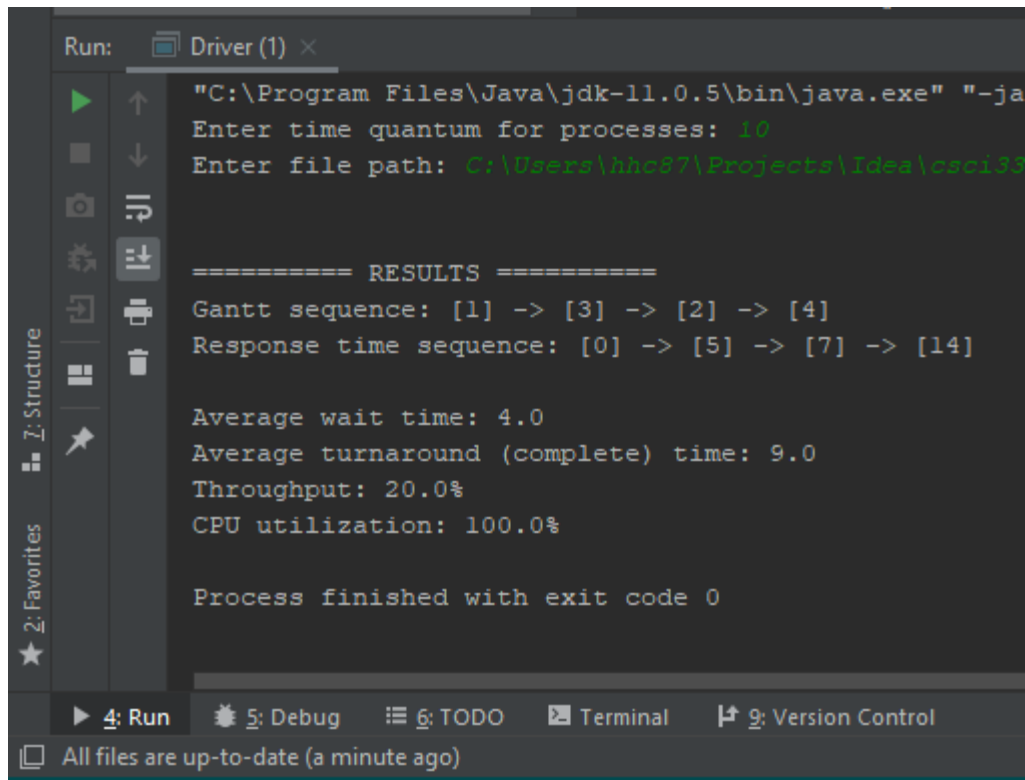
Average wait time: 4.0
Average turnaround (complete) time: 9.0
Throughput: 20.0%
CPU utilization: 100.0%

Process finished with exit code 0

4: Run 5: Debug 6: TODO Terminal 9: Version Control 0:
Build completed successfully in 1 s 790 ms (moments ago)
```

From the screenshot, we can see we have very good average wait time, and a bit better average turnaround time. Although the throughput still remaining 20%, the CPU utilization is overhead to 100%. Therefore, this is the desirable result that we are looking for. Also, from the Gantt sequence, we can see that all processes are instantly run once off on the CPU, but the response time increase. If the response time increase, then we lost the meaning of Round Robin's advantage.

iv. Time quantum = 10



```
Run: Driver (1) x
"C:\Program Files\Java\jdk-11.0.5\bin\java.exe" "-ja
Enter time quantum for processes: 10
Enter file path: C:\Users\hnc87\Projects\Idea\csci333\src\RoundRobin\input.txt

===== RESULTS =====
Gantt sequence: [1] -> [3] -> [2] -> [4]
Response time sequence: [0] -> [5] -> [7] -> [14]

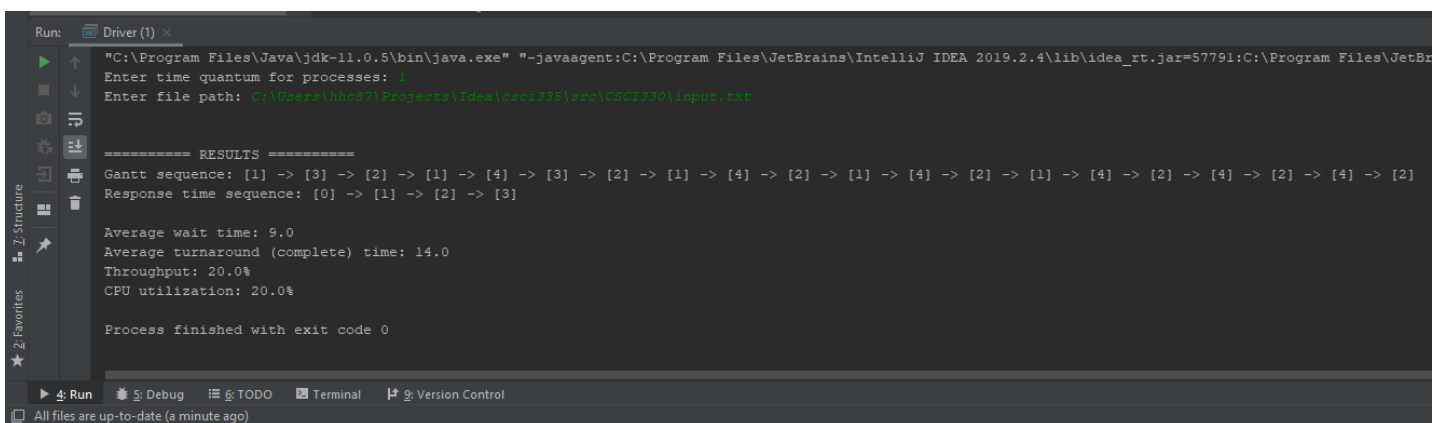
Average wait time: 4.0
Average turnaround (complete) time: 9.0
Throughput: 20.0%
CPU utilization: 100.0%

Process finished with exit code 0

4: Run 5: Debug 6: TODO Terminal 9: Version Control
All files are up-to-date (a minute ago)
```

From the screenshot, we can see that the result is the same as the time quantum equal to eight. Therefore, this suggested any time quantum greater or equal to eight has the same result, and we lost the advantage of Round Robin's fast response time.

v. Time quantum = 1



```
Run: Driver (1) x
"C:\Program Files\Java\jdk-11.0.5\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2019.2.4\lib\idea_rt.jar=57791:C:\Program Files\JetBr
Enter time quantum for processes: 1
Enter file path: C:\Users\hnc87\Projects\Idea\csci333\src\RoundRobin\input.txt

===== RESULTS =====
Gantt sequence: [1] -> [3] -> [2] -> [1] -> [4] -> [3] -> [2] -> [1] -> [4] -> [2] -> [1] -> [4] -> [2] -> [1] -> [4] -> [2] -> [4] -> [2]
Response time sequence: [0] -> [1] -> [2] -> [3]

Average wait time: 9.0
Average turnaround (complete) time: 14.0
Throughput: 20.0%
CPU utilization: 20.0%

Process finished with exit code 0

4: Run 5: Debug 6: TODO Terminal 9: Version Control
All files are up-to-date (a minute ago)
```

(since we already know any time quantum ≥ 8 will give the same result, so we set time quantum = 1)

From the screenshot, we can see that this program definitely overhead in context switch, but this time it gives perfect response time amongst the

processes. However, this program has very long average waiting time, and turnaround time. Therefore, we can conclude that the less the time quantum is, the higher average wait, turnaround time, and the number of context switches have. Although the throughput still remaining 20%, the CPU utilization is 20%, which is not what we try to achieve in CPU scheduling algorithm because it shows that we only used 20% of the CPU while the rest is idle.