#uni/semester3/Betriebssysteme/chapter7/simulator_fragen

1. Compute the response time and turnaround time when running three jobs of length 200 with the SJF and FIFO schedulers.

➜ FIFO:

```
vladb@VladB ~/G/h/S/B/H/HW3-Scheduler (master)> ./scheduler.py -p FIFO -l
200,200,200
ARG policy FIFO
ARG jlist 200,200,200

Here is the job list, with the run time of each job:
  Job 0 ( length = 200.0 )
  Job 1 ( length = 200.0 )
  Job 2 ( length = 200.0 )



Compute the turnaround time, response time, and wait time for each job.
When you are done, run this program again, with the same arguments,
but with -c, which will thus provide you with the answers. You can use
-s <somenumber> or your own job list (-l 10,15,20 for example)
to generate different problems for yourself.
```

Response time: (die Zeit zwischen Ankommen und erstes Starten)

- ➜ Job 0: 0 secs
- ➜ Job 1: 200 secs
- ➜ Job 2: 400 secs
- ➜ Average: 200 secs

Turnaround time: (die Zeit zwischen Ankommen und Ende des Jobs)

- ➜ Job 0: 200 secs
- ➜ Job 1: 400 secs
- ➜ Job 2: 600 secs
- ➜ Average: 400 secs

Wait time: (die Zeit während Job ready war)

- ➜ Job 0: 0 secs
- ➜ Job 1: 200 secs
- ➜ Job 2: 400 secs

➜ Average: 200 secs

```
vladb@VladB ~/G/h/S/B/H/HW3-Scheduler (master)> ./scheduler.py -p FIFO -l
200,200,200 -c
ARG policy FIFO
ARG jlist 200,200,200

Here is the job list, with the run time of each job:
  Job 0 ( length = 200.0 )
  Job 1 ( length = 200.0 )
  Job 2 ( length = 200.0 )


** Solutions **

Execution trace:
  [ time   0 ] Run job 0 for 200.00 secs ( DONE at 200.00 )
  [ time 200 ] Run job 1 for 200.00 secs ( DONE at 400.00 )
  [ time 400 ] Run job 2 for 200.00 secs ( DONE at 600.00 )

Final statistics:
  Job   0 -- Response: 0.00  Turnaround 200.00  Wait 0.00
  Job   1 -- Response: 200.00  Turnaround 400.00  Wait 200.00
  Job   2 -- Response: 400.00  Turnaround 600.00  Wait 400.00

  Average -- Response: 200.00  Turnaround 400.00  Wait 200.00
```

SJF:

```
vladb@VladB ~/G/h/S/B/H/HW3-Scheduler (master)> ./scheduler.py -p SJF -l
200,200,200
ARG policy SJF
ARG jlist 200,200,200

Here is the job list, with the run time of each job:
  Job 0 ( length = 200.0 )
  Job 1 ( length = 200.0 )
  Job 2 ( length = 200.0 )
```

Gleiche Werten wie FIFO, weil die Tasks gleich sind, also erster Job in der Liste wird auch als erstes ausgeführt.

2. Now do the same but with jobs of different lengths: 100, 200, and 300.

   FIFO:

```
vladb@VladB ~/G/h/S/B/H/HW3-Scheduler (master)> ./scheduler.py -p FIFO -l
100,200,300
ARG policy FIFO
ARG jlist 100,200,300

Here is the job list, with the run time of each job:
  Job 0 ( length = 100.0 )
  Job 1 ( length = 200.0 )
  Job 2 ( length = 300.0 )
```

   Response time:

   - ➜ Job 0: 0 secs
   - ➜ Job 1: 100 secs
   - ➜ Job 2: 300 secs
   - ➜ Average: 133.33 secs

   Turnaround time:

   - ➜ Job 0: 100 secs
   - ➜ Job 1: 300 secs
   - ➜ Job 2: 600 secs
   - ➜ Average: 333.33 secs

   Wait time:

   - ➜ Job 0: 0 secs
   - ➜ Job 1: 100 secs
   - ➜ Job 2: 300 secs
   - ➜ Average: 133.33 secs

   SJF:

```
vladb@VladB ~/G/h/S/B/H/HW3-Scheduler (master)> ./scheduler.py -p SJF -l
100,200,300
ARG policy SJF
```

```
ARG jlist 100,200,300

Here is the job list, with the run time of each job:
  Job 0 ( length = 100.0 )
  Job 1 ( length = 200.0 )
  Job 2 ( length = 300.0 )
```

Wider die gleichen Werte, da die Liste aufsteigend sortiert ist. Würde man die Reihenfolge ändern, würde sich das gleiche ergeben, wie FIFO 100, 200, 300

3. Now do the same, but also with the RR scheduler and a time-slice of 1.

```
vladb@VladB ~/G/h/S/B/H/HW3-Scheduler (master)> ./scheduler.py -p RR -l
200,200,200 -q 1
ARG policy RR
ARG jlist 200,200,200

Here is the job list, with the run time of each job:
  Job 0 ( length = 200.0 )
  Job 1 ( length = 200.0 )
  Job 2 ( length = 200.0 )
```

➜ RR mit Jobs = 3, Länge = 200 und Time slice = 1:

	0	1	2	3	4	5	6	7	8	9
J1:	x			x						
J2:		x			x					
J3:			x			x				

Response time:

- ➜ Job 0: 0 secs
- ➜ Job 1: 1 secs
- ➜ Job 2: 2 secs
- ➜ Average: 1 secs

Turnaround time:

Zeit zwischen zwei slices, in der job ready ist:

J0: 2

J1: 2

J2: 2

Anzahl Slices: (Job-Länge / slice-Länge):

J0: 200

J1: 200

J2: 200

Anzahl Stücke, wo Job wartet (Job-Länge - 1):

J0: 199

J1: 199

J2: 199

Ergebnisse:

J0: 0 + 200 + 199 * 2 = 598 secs

J1: 1 + J0 = 599 secs

J2: 2 + J0 = 600 secs

Average: 599 secs

Wait time:

➜ Job 0: 199 * 2 = 398 secs **oder schlauer: Turnaround - Runtime**

➜ Job 1: 199 * 2 + 1 = 399 secs

➜ Job 2: 199 * 2 + 2 = 400 secs

➜ Average: 399 secs

➜ RR mit Jobs = 100, 200, 300 und Time slice = 1:

Response time:

J0: 0 secs

J1: 1 secs

J2: 2 secs

Average: 1 secs

Turnaround time:

J0: (0 + 100 + 99 * 2) = 298 secs. (Die Rechnung ist vom erstes Laufen bis zum Schluss)

J1: 2 + 298 + (0 + 100 + 99 * 1) = 499 secs. (+2 weil 1 am Anfang und 1 am Ende von J0)

J2: 1 + 499 + 100 = 600 secs

Average: 465.67 secs

Wait time:

J0: 298 - 100 = 198 secs

J1: 499 - 200 = 299 secs
J2: 600 - 300 = 300 secs
Average: 265.67 secs

4. For what types of workloads does SJF deliver the same turnaround times as FIFO?
   ➜ Für die workloads wo alle Jobs die gleiche Länge haben und wenn bei FIFO die Jobs aufsteigend sortiert sind.
   ➜ Operating Systems 2014F: Assignment 3 - Soma-notes

5. For what types of workloads and quantum lengths does SJF deliver the same response times as RR?
   ➜ Wenn alle Jobs die gleiche Länge haben und die time slice Länge auch gleich der Job-Länge ist.
   ➜ Wenn bei N Jobs, die ersten N - 1, die time slice Länge haben, und die letzte Job-Länge kann beliebig sein
   ➜ Operating Systems 2014F: Assignment 3 - Soma-notes

6. What happens to response time with SJF as job lengths increase? Can you use the simulator to demonstrate the trend?
   ➜ Natürlich dass die response time auch damit steigert.
   ➜ Wenn man alle Job-Längen mit einem Faktor multipliziert, dann erhöht sich auch die response time um diesen Faktor.
   ➜ Eigentlich kann der längste Job beliebig groß sein, weil es dann nicht mehr die response time beeinflussen kann
   ➜ Operating Systems 2014F: Assignment 3 - Soma-notes

```
vladb@VladB ~/G/h/S/B/H/HW3-Scheduler (master)> ./scheduler.py -p SJF -l 1,2,3
-c
Final statistics:
  Job   0 -- Response: 0.00  Turnaround 1.00  Wait 0.00
  Job   1 -- Response: 1.00  Turnaround 3.00  Wait 1.00
  Job   2 -- Response: 3.00  Turnaround 6.00  Wait 3.00

  Average -- Response: 1.33  Turnaround 3.33  Wait 1.33
```

```
vladb@VladB ~/G/h/S/B/H/HW3-Scheduler (master)> ./scheduler.py -p SJF -l
```

```
10,20,30 -c
Final statistics:
  Job   0 -- Response: 0.00  Turnaround 10.00  Wait 0.00
  Job   1 -- Response: 10.00  Turnaround 30.00  Wait 10.00
  Job   2 -- Response: 30.00  Turnaround 60.00  Wait 30.00

  Average -- Response: 13.33  Turnaround 33.33  Wait 13.33
```

```
vladb@VladB ~/G/h/S/B/H/HW3-Scheduler (master)> ./scheduler.py -p SJF -l
100,200,300 -c
Final statistics:
  Job   0 -- Response: 0.00  Turnaround 100.00  Wait 0.00
  Job   1 -- Response: 100.00  Turnaround 300.00  Wait 100.00
  Job   2 -- Response: 300.00  Turnaround 600.00  Wait 300.00

  Average -- Response: 133.33  Turnaround 333.33  Wait 133.33
```

➜ Wenn der zweit größte Job sich um eins erhöht, dann erhöht sich die average response time um 0,33 secs. (Bei 3 jobs)
➜ Wenn sich der kleinste Job um eins erhöht, dann erhöht sich die average Response time um 0,66 secs. (Bei 3 jobs)
➜ ALLGEMEINEREGEL: Zeiterhöhung = (1 / Anz. Jobs) * (Anz. Jobs - Rang)
Rang beschreibt die Position in der Liste, wenn alle Jobs aufsteigend sortiert sind. Also kürzeste Job, hat den Rang 1

```
vladb@VladB ~/G/h/S/B/H/HW3-Scheduler (master)> ./scheduler.py -p SJF -l
1,4,6,9 -c
Final statistics:
  Job   0 -- Response: 0.00  Turnaround 1.00  Wait 0.00
  Job   1 -- Response: 1.00  Turnaround 5.00  Wait 1.00
  Job   2 -- Response: 5.00  Turnaround 11.00  Wait 5.00
  Job   3 -- Response: 11.00  Turnaround 20.00  Wait 11.00

  Average -- Response: 4.25  Turnaround 9.25  Wait 4.25
```

Zeiterhöhung um 0.75 secs, nach der Formel

```
vladb@VladB ~/G/h/S/B/H/HW3-Scheduler (master)> ./scheduler.py -p SJF -l
2,4,6,9 -c
Final statistics:
  Job   0 -- Response: 0.00  Turnaround 2.00  Wait 0.00
  Job   1 -- Response: 2.00  Turnaround 6.00  Wait 2.00
  Job   2 -- Response: 6.00  Turnaround 12.00  Wait 6.00
  Job   3 -- Response: 12.00  Turnaround 21.00  Wait 12.00

  Average -- Response: 5.00  Turnaround 10.25  Wait 5.00
```

```
vladb@VladB ~/G/h/S/B/H/HW3-Scheduler (master)> ./scheduler.py -p SJF -l
2,5,6,9 -c
Final statistics:
  Job   0 -- Response: 0.00  Turnaround 2.00  Wait 0.00
  Job   1 -- Response: 2.00  Turnaround 7.00  Wait 2.00
  Job   2 -- Response: 7.00  Turnaround 13.00  Wait 7.00
  Job   3 -- Response: 13.00  Turnaround 22.00  Wait 13.00

  Average -- Response: 5.50  Turnaround 11.00  Wait 5.50
```

```
vladb@VladB ~/G/h/S/B/H/HW3-Scheduler (master)> ./scheduler.py -p SJF -l
2,5,7,9 -c
Final statistics:
  Job   0 -- Response: 0.00  Turnaround 2.00  Wait 0.00
  Job   1 -- Response: 2.00  Turnaround 7.00  Wait 2.00
  Job   2 -- Response: 7.00  Turnaround 14.00  Wait 7.00
  Job   3 -- Response: 14.00  Turnaround 23.00  Wait 14.00

  Average -- Response: 5.75  Turnaround 11.50  Wait 5.75
```

7. What happens to response time with RR as quantum lengths increase? Can you write an

equation that gives the worst-case response time, given N jobs?
➜ Time slices die größer gleich der Länge des größten Jobs liefern die gleichen Response times
➜ Erhöht man die time slices um einen Faktor, so multipliziert sich die Response time auch um diesen Faktor

```
vladb@VladB ~/G/h/S/B/H/HW3-Scheduler (master)>
./scheduler.py -p RR -l 100,200,300,400 -q 1 -c
Final statistics:
  Job   0 -- Response: 0.00  Turnaround 397.00  Wait 297.00
  Job   1 -- Response: 1.00  Turnaround 698.00  Wait 498.00
  Job   2 -- Response: 2.00  Turnaround 899.00  Wait 599.00
  Job   3 -- Response: 3.00  Turnaround 1000.00  Wait 600.00

  Average -- Response: 1.50  Turnaround 748.50  Wait 498.50
```

```
vladb@VladB ~/G/h/S/B/H/HW3-Scheduler (master)>
./scheduler.py -p RR -l 100,200,300,400 -q 10 -c
Final statistics:
  Job   0 -- Response: 0.00  Turnaround 370.00  Wait 270.00
  Job   1 -- Response: 10.00  Turnaround 680.00  Wait 480.00
  Job   2 -- Response: 20.00  Turnaround 890.00  Wait 590.00
  Job   3 -- Response: 30.00  Turnaround 1000.00  Wait 600.00

  Average -- Response: 15.00  Turnaround 735.00  Wait 485.00
```

```
vladb@VladB ~/G/h/S/B/H/HW3-Scheduler (master)>
./scheduler.py -p RR -l 100,200,300,400 -q 100 -c
Final statistics:
  Job   0 -- Response: 0.00  Turnaround 100.00  Wait 0.00
  Job   1 -- Response: 100.00  Turnaround 500.00  Wait 300.00
  Job   2 -- Response: 200.00  Turnaround 800.00  Wait 500.00
  Job   3 -- Response: 300.00  Turnaround 1000.00  Wait 600.00
```

```
  Average -- Response: 150.00  Turnaround 600.00  Wait 350.00
```

➜ Erhöht man die time slices um 1, dann erhöht sich time Response um eine festen Zahl. Diese Zahl ergibst sich aus der Gesamtverzögerung (relativ zu time slice - 1) geteilt durch Anzahl Jobs

```
vladb@VladB ~/G/h/S/B/H/HW3-Scheduler (master)>
./scheduler.py -p RR -l 100,200,300,400 -q 10 -c
Final statistics:
  Job   0 -- Response: 0.00  Turnaround 370.00  Wait 270.00
  Job   1 -- Response: 10.00  Turnaround 680.00  Wait 480.00
  Job   2 -- Response: 20.00  Turnaround 890.00  Wait 590.00
  Job   3 -- Response: 30.00  Turnaround 1000.00  Wait 600.00

  Average -- Response: 15.00  Turnaround 735.00  Wait 485.00
```

```
vladb@VladB ~/G/h/S/B/H/HW3-Scheduler (master)>
./scheduler.py -p RR -l 100,200,300,400 -q 11 -c
Final statistics:
  Job   0 -- Response: 0.00  Turnaround 397.00  Wait 297.00
  Job   1 -- Response: 11.00  Turnaround 696.00  Wait 496.00
  Job   2 -- Response: 22.00  Turnaround 897.00  Wait 597.00
  Job   3 -- Response: 33.00  Turnaround 1000.00  Wait 600.00

  Average -- Response: 16.50  Turnaround 747.50  Wait 497.50
```

➜ Mit time slice Länge wird die response time auch größer, da die Zeit die ein Job auf sein erstes time slice warten muss, ist proportional mit der Anzahl der Jobs und der time slice Länge. Also in einem System mit N Jobs und einer time slice Länge von T, ein Job (in worst case) muss T*(N-1) secs warten. Worst case ist dann der letzte Job!
Worst case wenn quantum die gleiche Länge wie
Response time im worst case = (qlenght * sum(n-1))/ n
sum(n-1) ➜ eigentlich von 0 bis zum vorletzten

➜ Operating Systems 2014F: Assignment 3 - Soma-notes