#uni/semester3/Betriebssysteme/chapter8/simulator-fragen

1. Run a few randomly-generated problems with just two jobs and two queues; compute the MLFQ execution trace for each. Make your life easier by limiting the length of each job and turning off I/Os.
   ➜ Alle Jobs werden auf der Queue mit der größten Priorität am Anfang gesetzt. Dann läuft der erste um die Länge eines time slices und seine Priorität wird erniedrigt. Dann macht man das gleiche mit dem nächsten Job. Die übrigen Teile der Jobs werden dann auf der nächsten Queue
   ausgeführt

```
vladb@VladB ~/G/h/S/B/H/HW4-MLFQ (master)> ./mlfq.py -j 2 -n 2 -m 20 -M 0 -s 5
-c
Here is the list of inputs:
OPTIONS jobs 2
OPTIONS queues 2
OPTIONS allotments for queue  1 is   1
OPTIONS quantum length for queue  1 is  10
OPTIONS allotments for queue  0 is   1
OPTIONS quantum length for queue  0 is  10
OPTIONS boost 0
OPTIONS ioTime 5
OPTIONS stayAfterIO False
OPTIONS iobump False


For each job, three defining characteristics are given:
  startTime : at what time does the job enter the system
  runTime   : the total CPU time needed by the job to finish
  ioFreq    : every ioFreq time units, the job issues an I/O
              (the I/O takes ioTime units to complete)

Job List:
  Job  0: startTime   0 - runTime  12 - ioFreq   0
  Job  1: startTime   0 - runTime  16 - ioFreq   0


Execution Trace:
```

```
[ time 0 ] JOB BEGINS by JOB 0
[ time 0 ] JOB BEGINS by JOB 1
[ time 0 ] Run JOB 0 at PRIORITY 1 [ TICKS 9 ALLOT 1 TIME 11 (of 12) ]
[ time 1 ] Run JOB 0 at PRIORITY 1 [ TICKS 8 ALLOT 1 TIME 10 (of 12) ]
[ time 2 ] Run JOB 0 at PRIORITY 1 [ TICKS 7 ALLOT 1 TIME 9 (of 12) ]
[ time 3 ] Run JOB 0 at PRIORITY 1 [ TICKS 6 ALLOT 1 TIME 8 (of 12) ]
[ time 4 ] Run JOB 0 at PRIORITY 1 [ TICKS 5 ALLOT 1 TIME 7 (of 12) ]
[ time 5 ] Run JOB 0 at PRIORITY 1 [ TICKS 4 ALLOT 1 TIME 6 (of 12) ]
[ time 6 ] Run JOB 0 at PRIORITY 1 [ TICKS 3 ALLOT 1 TIME 5 (of 12) ]
[ time 7 ] Run JOB 0 at PRIORITY 1 [ TICKS 2 ALLOT 1 TIME 4 (of 12) ]
[ time 8 ] Run JOB 0 at PRIORITY 1 [ TICKS 1 ALLOT 1 TIME 3 (of 12) ]
[ time 9 ] Run JOB 0 at PRIORITY 1 [ TICKS 0 ALLOT 1 TIME 2 (of 12) ]
[ time 10 ] Run JOB 1 at PRIORITY 1 [ TICKS 9 ALLOT 1 TIME 15 (of 16) ]
[ time 11 ] Run JOB 1 at PRIORITY 1 [ TICKS 8 ALLOT 1 TIME 14 (of 16) ]
[ time 12 ] Run JOB 1 at PRIORITY 1 [ TICKS 7 ALLOT 1 TIME 13 (of 16) ]
[ time 13 ] Run JOB 1 at PRIORITY 1 [ TICKS 6 ALLOT 1 TIME 12 (of 16) ]
[ time 14 ] Run JOB 1 at PRIORITY 1 [ TICKS 5 ALLOT 1 TIME 11 (of 16) ]
[ time 15 ] Run JOB 1 at PRIORITY 1 [ TICKS 4 ALLOT 1 TIME 10 (of 16) ]
[ time 16 ] Run JOB 1 at PRIORITY 1 [ TICKS 3 ALLOT 1 TIME 9 (of 16) ]
[ time 17 ] Run JOB 1 at PRIORITY 1 [ TICKS 2 ALLOT 1 TIME 8 (of 16) ]
[ time 18 ] Run JOB 1 at PRIORITY 1 [ TICKS 1 ALLOT 1 TIME 7 (of 16) ]
[ time 19 ] Run JOB 1 at PRIORITY 1 [ TICKS 0 ALLOT 1 TIME 6 (of 16) ]
[ time 20 ] Run JOB 0 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 1 (of 12) ]
[ time 21 ] Run JOB 0 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 0 (of 12) ]
[ time 22 ] FINISHED JOB 0
[ time 22 ] Run JOB 1 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 5 (of 16) ]
[ time 23 ] Run JOB 1 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 4 (of 16) ]
[ time 24 ] Run JOB 1 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 3 (of 16) ]
[ time 25 ] Run JOB 1 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 2 (of 16) ]
[ time 26 ] Run JOB 1 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 1 (of 16) ]
[ time 27 ] Run JOB 1 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 0 (of 16) ]
[ time 28 ] FINISHED JOB 1

Final statistics:
  Job  0: startTime   0 - response   0 - turnaround  22
  Job  1: startTime   0 - response  10 - turnaround  28

  Avg  1: startTime n/a - response 5.00 - turnaround 25.00
```

2. How would you run the scheduler to reproduce each of the examples in the chapter?

   Example 1:

```
vladb@VladB ~/G/h/S/B/H/HW4-MLFQ (master)> ./mlfq.py -l 0,200,0 -n 3 -c
```

   Example 2:

```
vladb@VladB ~/G/h/S/B/H/HW4-MLFQ (master)> ./mlfq.py -l 0,180,0:100,20,0 -n 3 -
c
```

   Example 3:

```
./mlfq.py -l 0,175,0:50,25,1 -n 3 -S -i 5 -c
```

   Für den Rest der Figures siehe https://github.com/xxyzz/ostep-hw/tree/master/8

3. How would you configure the scheduler parameters to behave just like a round-robin scheduler?

   ➜ Nur eine queue zu haben mit einer festen time slice.

```
vladb@VladB ~/G/h/S/B/H/HW4-MLFQ (master)> ./mlfq.py -l 0,100,0:0,50,0 -n 1 -c
```

   time slice <= (max job length / jobs number) mit mehreren Queues aber die Reihenfolge dann wie RR

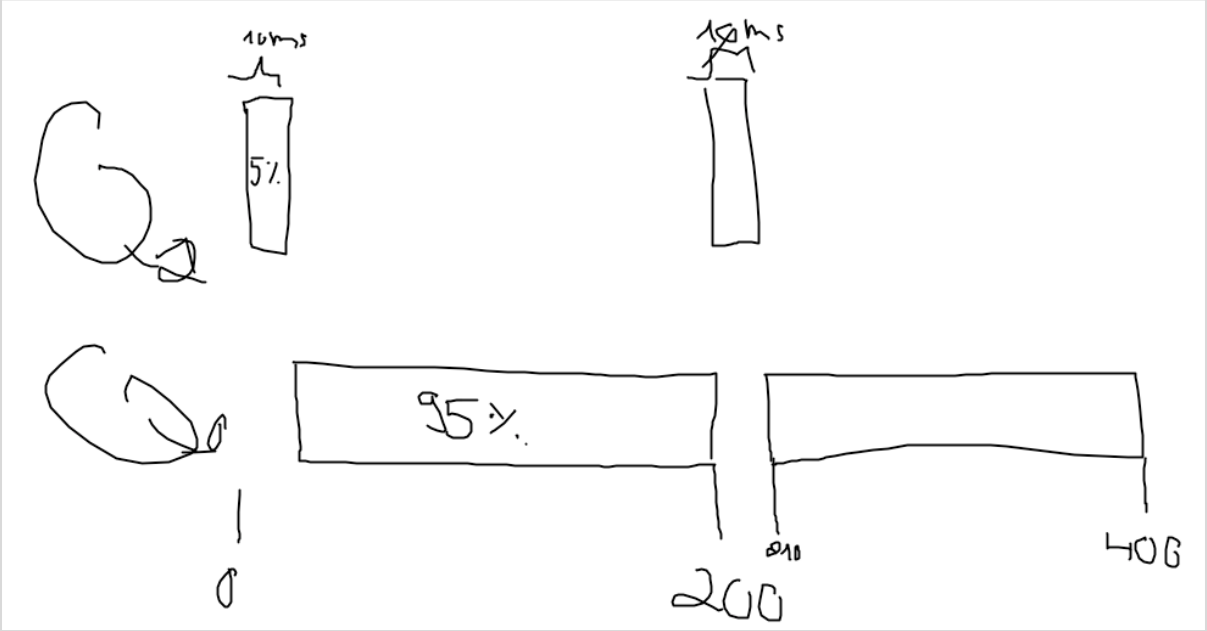
4. Craft a workload with two jobs and scheduler parameters so that one job takes advantage of the older Rules 4a and 4b (turned on with the -S flag) to game the scheduler and obtain 99% of the CPU over a particular time interval.

```
vladb@VladB ~/G/h/S/B/H/HW4-MLFQ (master)>
./mlfq.py -S -i 1 -l 0,297,99:0,60,0 -q 100 -n 3 -I -c
```

   Oder

```
-l 0,1000,99:0,1000,0 -q 100 -i 1 -S
```

5. Given a system with a quantum length of 10 ms in its highest queue, how often would you have to boost jobs back to the highest priority level (with the -B flag) in order to guarantee that a single long-running (and potentially-starving) job gets at least 5% of the CPU?
   ➜ Wenn 10ms , 5% sind, dann ist 100%, 200 ms. Also alle 200 ms



6. One question that arises in scheduling is which end of a queue to add a job that just finished I/O; the -I flag changes this behavior for this scheduling simulator. Play around with some workloads and see if you can see the effect of this flag
   ➜ Vergleiche

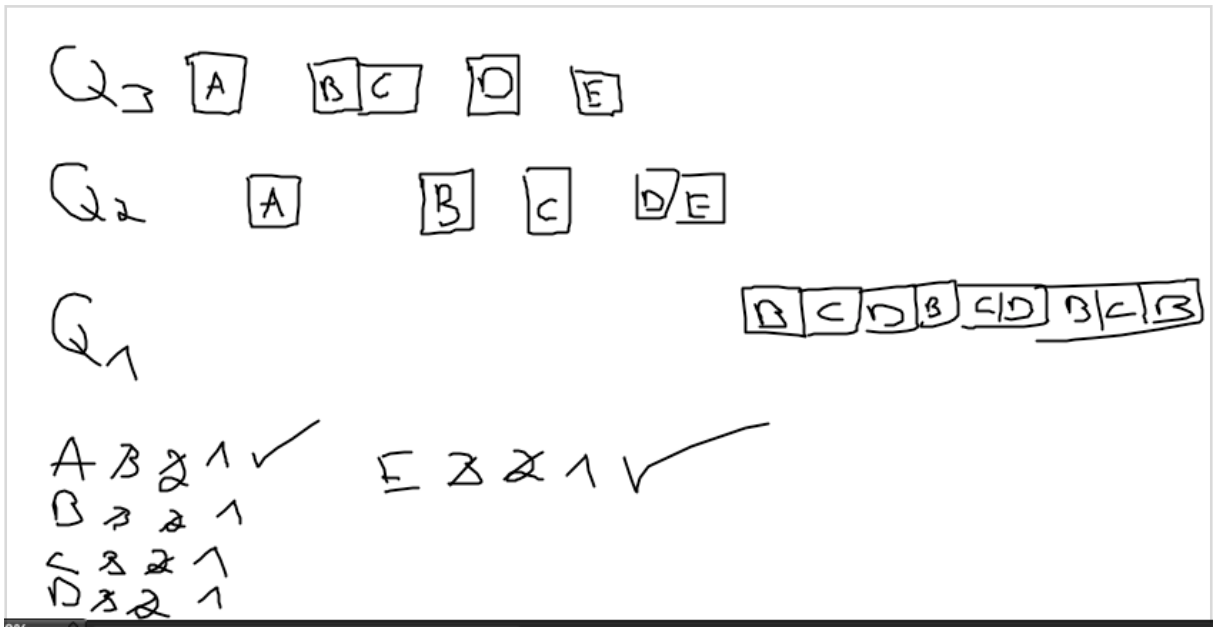
```
./mlfq.py -l 0,70,30:29,50,0 -i 1 -n 1 -c
```

Mit:

```
./mlfq.py -l 0,70,30:29,50,0 -i 1 -n 1 -I -c
```

Der erste Job läuft für 30 Ticks, dann startet IO. Währenddessen läuft der zweite Job bis er den time slice macht und dann läuft der erste Job wieder. Mit -I wird der zweite nur einmal ausgeführt, und bei Rückkehr von IO macht der erste wieder den time slice. Von dort an wechseln sie sich ab, bis der nächste IO gemacht wird.

Frage zur Folie 13:



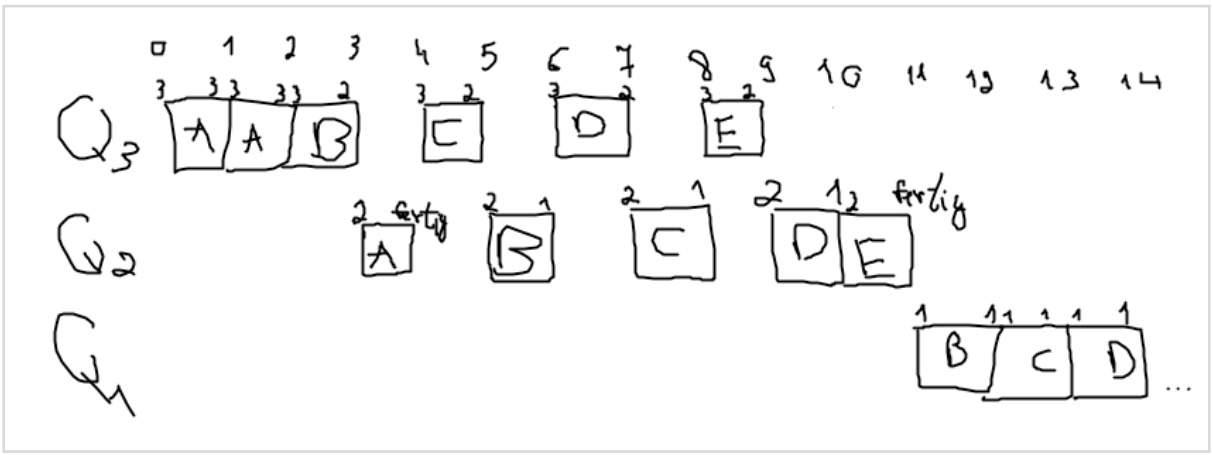
➜ Vergessen am Zeitpunkt 3 (Vorsicht es fängt mit Zeitpunkt 0) sollte A noch einmal ausgeführt werden. In welche queue A dann ist, siehe RIOT chat:

```
wieso ist A nicht in q3 nach den ersten zwei  durchläufen
Bei dieser speziellen Änderung des MLFQ (siehe auch aktualisiere Folien) Nur
wenn Task verdrängt wird, nicht wenn 'nur' Zeitscheibe rum ist
```

t	run	q1	q2	q3	
0	A				
1	A				
2	B		A		
3	A		B		
4	C		B		A ready
5	B		C		
6	D		C	B	
7	C		D	B	
8	E		D	B,C	
9	D		E	B,C	

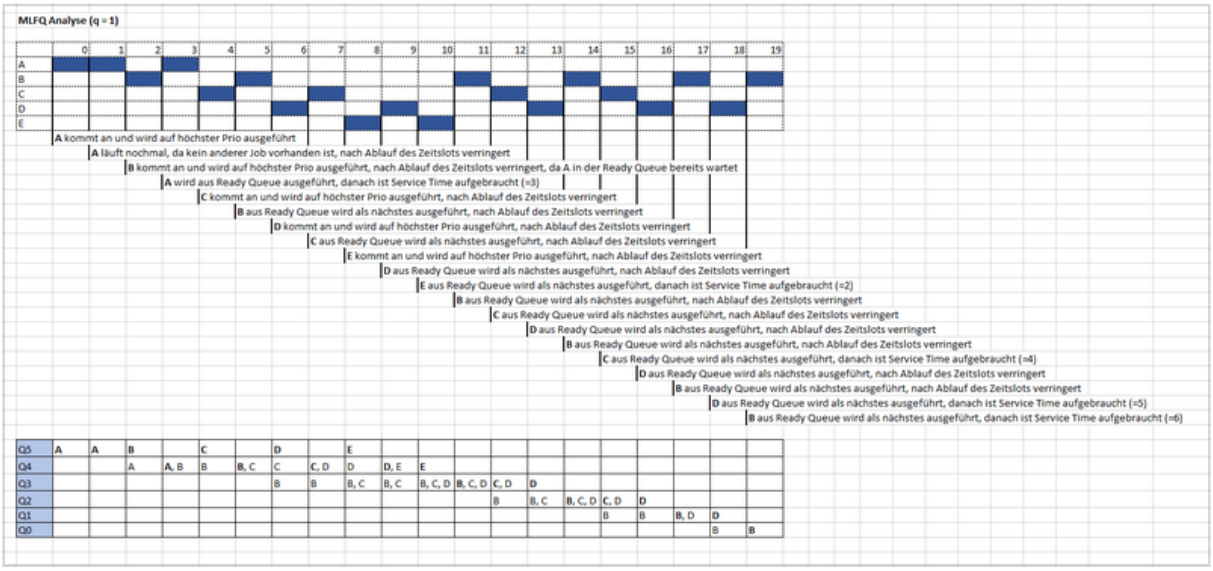
Check if other tasks in system are ready, then move task to lower priority queue!

Wenn kein anderer Job in system, dann Priorität NICHT um 1 verkleinern. Deshalb wird A in 0 und 1 nicht verkleinert. Aber in B wird es verkleinert. Im Schritt 3 stehen sie auf gleiche Ebene, deshalb wird A ausgeführt!



Bei MLFQ Default ist dass ein Job immer hintern angestellt wird! Bei E/A jobs ist so ne Implementierungssache ob es nach vorne gestellt werden. Das gleiche mit boost, nur die unterste queue wird geboostet, in der Reihenfolge die die jobs waren, nach hinten in der

ersten.



Q5	A	A	B		C		D		E										
Q4			A	A, B	B	B, C	C	C, D	D	D, E	E								
Q3							B	B	B, C	B, C	B, C, D	B, C, D	C, D	D					
Q2												B	B, C	B, C, D	C, D	D			
Q1															B	B	B, D	D	
Q0																		B	B