Julia Liszniańska
242567

Exercise 1

In the exercise I had to compare the average waiting time for the processes involved for different scheduling algorithms (FCFS, non pre-emptive SJF, pre-emptive SJF and Robin Round).
I created a test file with 40 processes with different arrival times:

Process ID	Arrival time	Burst time
1	3	5
2	4	6
3	1	7
4	2	8
5	7	2
6	0	4
7	5	1
8	1	6
9	7	8
10	8	5
11	9	3
12	3	2
13	2	1
14	1	6
15	6	9
16	4	4
17	5	2
18	3	8
19	1	9
20	3	5
21	2	7
22	1	9
23	4	2
24	1	7
25	4	5
26	2	6
27	2	1
28	6	8
29	4	7
30	3	8
31	5	9
32	7	1
33	8	3

34	1	1
35	1	3
36	5	5
37	3	6
38	7	7
39	8	2
40	2	6

Data set 1

and also a smaller data set with 5 different processes:

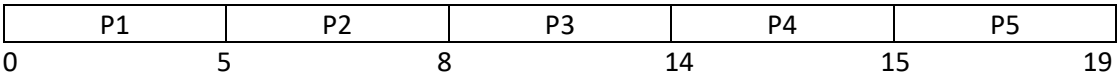
Process ID	Arrival time	Burst time
1	0	5
2	1	3
3	3	6
4	5	1
5	6	4

Data set 2

1. **First Come First Serve**

   First Come First Serve, is just like FIFO(First in First out) Queue data structure, where the data element which is added to the queue first, is the one who leaves the queue first. I sorted my list according to the arrival time and then calculated average time, which is waiting time for the last element divided by number of all processes in the data set.
   For the data set 1 the average time was 104.65, and for the data set 2 it was 7.4.

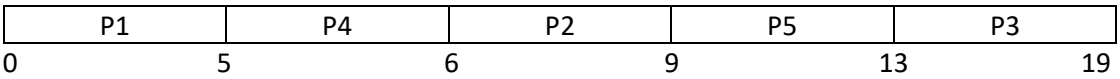
   For data set 2 the queue will look like that:



2. **Non Pre-emptive Shortest Job First**

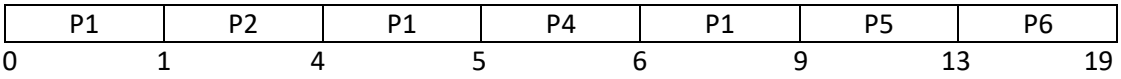
   Shortest Job First scheduling works on the process with the shortest burst time or duration first. In simple words, this algorithm executes processes with their increasing order of CPU burst time. If all processes are having same CPU burst time then this works as FCFS algorithm. SJF minimizes average waiting time, however it's hard to predict CPU burst time in advance.
   In order to apply this algorithm, I sorted the array by arrival time and then for every process I checked if the arrival time is smaller or equal a current time. I saved the processes which fulfilled this condition in temporary array and sorted them accordingly to their burst time.
   For bigger data set the average waiting time was 66.525 and for the smaller one it was only 3.6.

   For data set 2 the queue will look like that:



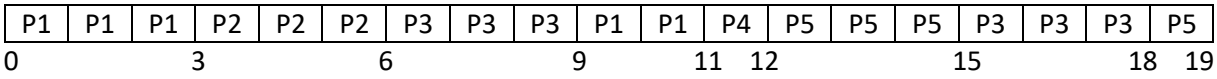
3. **Pre-emptive Shortest Job First (or Shortest Remaining Time First)**
   In Preemptive Shortest Job First Scheduling, jobs are put into ready queue as they arrive, but as a process with short burst time arrives, the existing process is preempted or removed from execution, and the shorter job is executed first.
   Again, I sorted the array accordingly to the arrival time and burst time. Until all processes got completely executes I was finding process with minimum remaining time at every single time lap, reduced its time by 1 and checked if its remaining time became 0.
   For data set 1 the average time was 97.65 and for second data set it was 6.0.

   For data set 2 the queue will look like that:



4. **Robin-Round**
   Round robin scheduling algorithm is one of the important scheduling algorithm in job scheduling. It is the preemptive scheduling algorithm. Round robin uses time slice (fixed time period) for execution of the process, called time quantum. In RR all the processes have the equal priority because of fixed time quantum.
   To schedule processes fairly, a round-robin scheduler generally employs time-sharing, giving each job a time slot or quantum, and interrupting the job if it is not completed by then. The job is resumed next time a time slot is assigned to that process. If the process terminates or changes its state to waiting during its attributed time quantum, the scheduler selects the first process in the ready queue to execute.
   For quantum equal to 3, the average waiting time for the bigger array was 128.125 and for smaller 9.4.

   For data set 2 the queue will look like that:



   As we can see, the fastest algorithm is non pre-emptive Short Job First scheduling algorithm and the slowest one is robin-round. All of the above algorithms have both advantages and disadvantages. In my opinion, the best algorithm is non pre-emptive Shortest Job First because it is easy to implement and it works fast.