‘CPU scheduling algorithms manual’

1.non-preemptive

- *FCFS*
- *SJF*
- *PS*

2.Preemptive

- *RR*
- *SRTF*
- *EDF*

Each algorithm is put in its own separate file, the reason for this decision is; different data structures are used for two of them with respect to the others. More specifically, FCFS and SJF are implemented thinking of processes as objects with properties of unique id, arrival time, burst time while others use lists of lists(matrices) to represent processes.

Requirements:

All programs are written in python, so you must have python installed (3.0>). In addition, you will also need to install “numpy” library, which is used to ease the task of opening files. (Nevertheless, it’s a useful library in general, so it’s a good idea to have it in your arsenal!)

To install “numpy”, use one of the following:

LINUX

sudo apt install python3-pip

pip3 install numpy

WINDOWS

pip install numpy

How to use:

All the programs take a file containing the information about the processes as command line argument.

Given the fact that all programs use different sets of information, individual examples are given for each following a brief explanation.

The lines can contain extra spaces at the end, but no extra letter. The file can contain empty new lines in between.

- FCFS
  “The format of each line of the input file is; | id | arrival time | burst time |”
  “The program will return a chart of the processes, a waiting time chart, Gantt diagram and a number representing the average waiting time on the output terminal.”
  “This program in particular uses a FIFO data structure, taken from the python collections library. While this algorithm is by far the simplest among them all, and can be implemented in very few lines, I

implemented it this way just to simulate as close as possible the interaction of the processes and the CPU, where they are put in a queue during which period their waiting time is calculated, then they get dequeued at the time that the CPU is ready for them."

<u>How to run</u>:

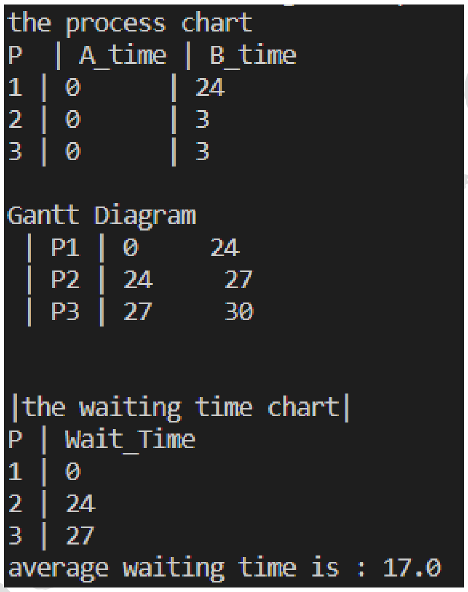
<u>-windows</u>

```
py FCFS.py <name_of_the_file_containing_processes>
```

<u>-linux</u>

```
python3 FCFS.py <name_of_the_file_containing_processes>
```

The output, given the processes in the file is:



```
the process chart
P  | A_time | B_time
1 | 0      | 24
2 | 0      | 3
3 | 0      | 3

Gantt Diagram
 | P1 | 0     24
 | P2 | 24     27
 | P3 | 27     30


|the waiting time chart|
P | Wait_Time
1 | 0
2 | 24
3 | 27
average waiting time is : 17.0
```

- SJF

  "The format of each line of the input file is; | id | arrival time | burst time |"

  "the program will return a chart of the processes, Gantt diagram and a number representing the average waiting time on the output terminal."

  <u>How to run</u>:

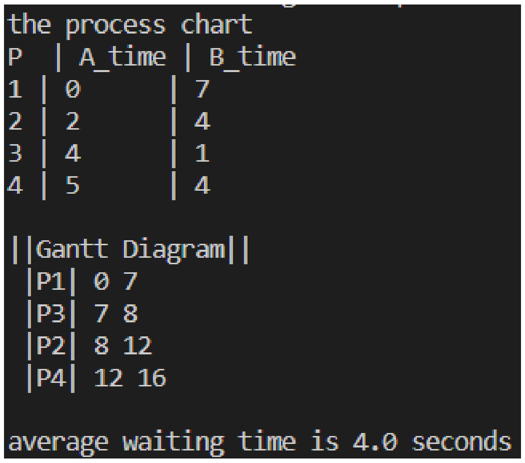
  <u>-windows</u>

```
py SJF.py <name_of_the_file_containing_processes>
```

<u>-linux</u>

```
python3 SJF.py <name_of_the_file_containing_processes>
```

The output, given the processes in the file is:



```
the process chart
P  | A_time | B_time
1 | 0      | 7
2 | 2      | 4
3 | 4      | 1
4 | 5      | 4

||Gantt Diagram||
 |P1| 0 7
 |P3| 7 8
 |P2| 8 12
 |P4| 12 16

average waiting time is 4.0 seconds
```

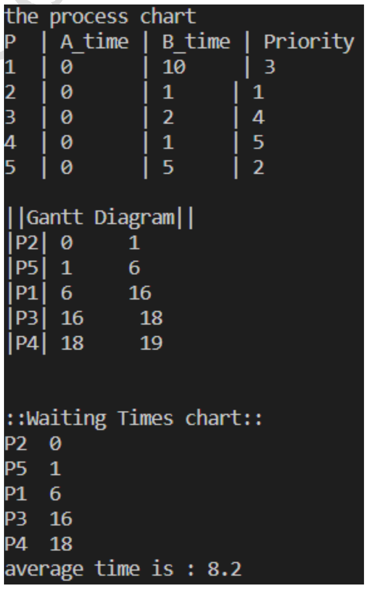
- PS
  "The format of each line of the input file is; | id | arrival time | burst time | priority |"
  "The program will return a chart of the processes, a waiting time chart, Gantt diagram and a number representing the average waiting time on the output terminal."
  How to run:
  -windows

  ```
  py PS.py <name_of_the_file_containing_processes>
  ```

  -linux

  ```
  python3 PS.py <name_of_the_file_containing_processes>
  ```

  The output, given the processes in the file is:



```
the process chart
P  | A_time | B_time | Priority
1  | 0      | 10     | 3
2  | 0      | 1     | 1
3  | 0      | 2     | 4
4  | 0      | 1     | 5
5  | 0      | 5     | 2

||Gantt Diagram||
|P2| 0     1
|P5| 1     6
|P1| 6     16
|P3| 16     18
|P4| 18     19


::Waiting Times chart::
P2  0
P5  1
P1  6
P3  16
P4  18
average time is : 8.2
```

- RR
  "The format of each line of the input file is; | id | arrival time | burst time |"
  "the program will return a chart of the processes, Gantt diagram and a number representing the average waiting time on the output terminal."
  How to run:

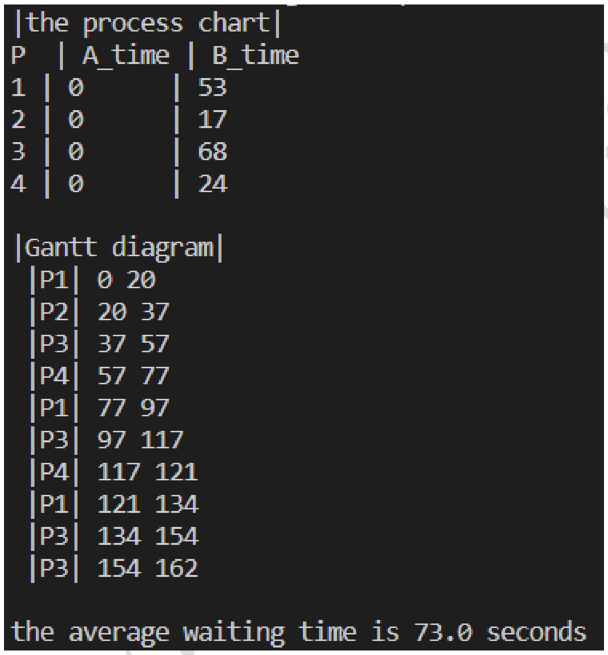
-windows

```
py PS.py <name_of_the_file_containing_processes> <quantum>
```

-linux

```
python3 PS.py <name_of_the_file_containing_processes> <quantum>
```

The output, given the processes in the file is:



```
|the process chart|
P  | A_time | B_time
1 | 0      | 53
2 | 0      | 17
3 | 0      | 68
4 | 0      | 24

|Gantt diagram|
 |P1| 0 20
 |P2| 20 37
 |P3| 37 57
 |P4| 57 77
 |P1| 77 97
 |P3| 97 117
 |P4| 117 121
 |P1| 121 134
 |P3| 134 154
 |P3| 154 162

the average waiting time is 73.0 seconds
```

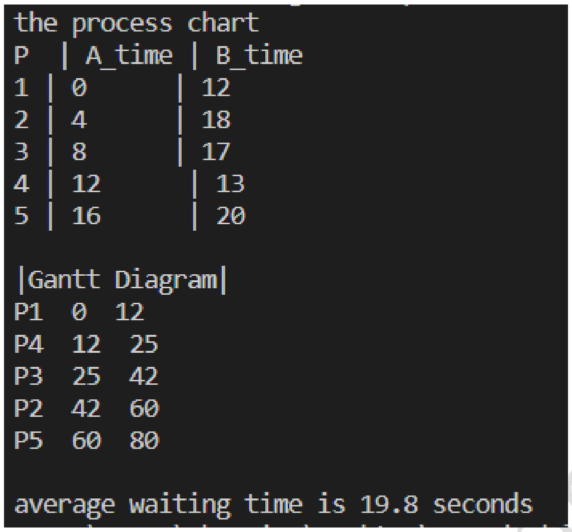
- SRTF
  "The format of each line of the input file is; | id | arrival time | burst time |"
  "the program will return a chart of the processes, Gantt diagram and a number representing the average waiting time on the output terminal."
  How to run:
  -windows

```
py SRTF.py <name_of_the_file_containing_processes>
```

-linux

```
python3 SRTF.py <name_of_the_file_containing_processes>
```

The output, given the processes in the file is:



```
the process chart
P   | A_time | B_time
1 | 0      | 12
2 | 4      | 18
3 | 8      | 17
4 | 12      | 13
5 | 16      | 20

|Gantt Diagram|
P1  0  12
P4  12  25
P3  25  42
P2  42  60
P5  60  80

average waiting time is 19.8 seconds
```

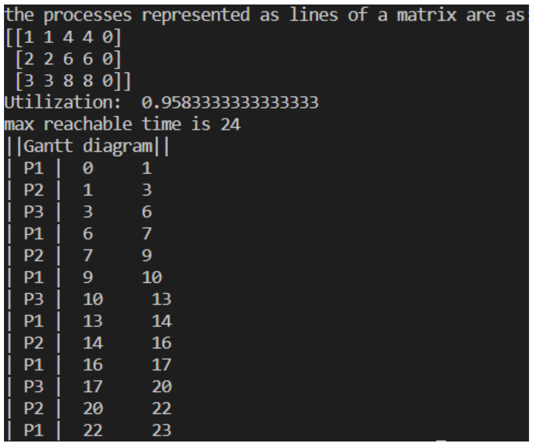
- EDF
  “The format of each line of the input file is; | id | execution time | period | deadline | arrival time |”
  “The program is written considering all the arrival times of the processes are equal to zero.”
  “It will print the following to the output terminal; the maximum reachable time, the utilization factor, gantt diagram representing the periods of the execution of the processes.”
  “Take into consideration that if the utilization factor results bigger than one, the processes are not feasible and the program will exit printing the corresponding message on the screen.”
  “In addition to numpy, python math library is also used.”
  How to run:
  -windows

```
py EDF.py <name_of_the_file_containing_processes>
```

-linux

```
python3 EDF.py <name_of_the_file_containing_processes>
```

The output, given the processes in the file is:



```
the processes represented as lines of a matrix are as
[[1 1 4 4 0]
 [2 2 6 6 0]
 [3 3 8 8 0]]
Utilization:  0.9583333333333333
max reachable time is 24
||Gantt diagram||
| P1 |  0     1
| P2 |  1     3
| P3 |  3     6
| P1 |  6     7
| P2 |  7     9
| P1 |  9     10
| P3 |  10     13
| P1 |  13     14
| P2 |  14     16
| P1 |  16     17
| P3 |  17     20
| P2 |  20     22
| P1 |  22     23
```

**In all the programs, enough comments are inserted to ease the understanding of how they work.*