Schedulers report

The folder includes 3 files

common.c - here I included some common functionality, and this is included in the other two files. It is directly imported as "common.c" file, which might not be conventional.

fcfs.c

sjf.c

To build and run either of them, do

```
gcc ./7_scheduling/fcfs.c -o ./build/fcfs && ./build/fcfs
gcc ./7_scheduling/sjf.c -o ./build/sjf && ./build/sjf
```

like you would for any other c program.

common.c

I needed to do a few things for both versions of the schedulers. I needed to sort with some logic, then calculate for each process the timings, also the average times, and print them somehow.

I included all this functionality in this file, along with a "run" function, that just accepts the function that will be used to sort the array as an argument. It calls the sort function, runs all the calculations, prints everything.

Initially I thought that I'd use quicksort for both my schedulers, that's why it's also included in the common.c file. It's a non-recursive implementation, so I don't keep pushing recursive calls to the stack. Instead I use an array that holds 2 ints for each process. One int is for the index of the pivot, and the next int is where the current range will end. So pivot is chosen the left-most element always.

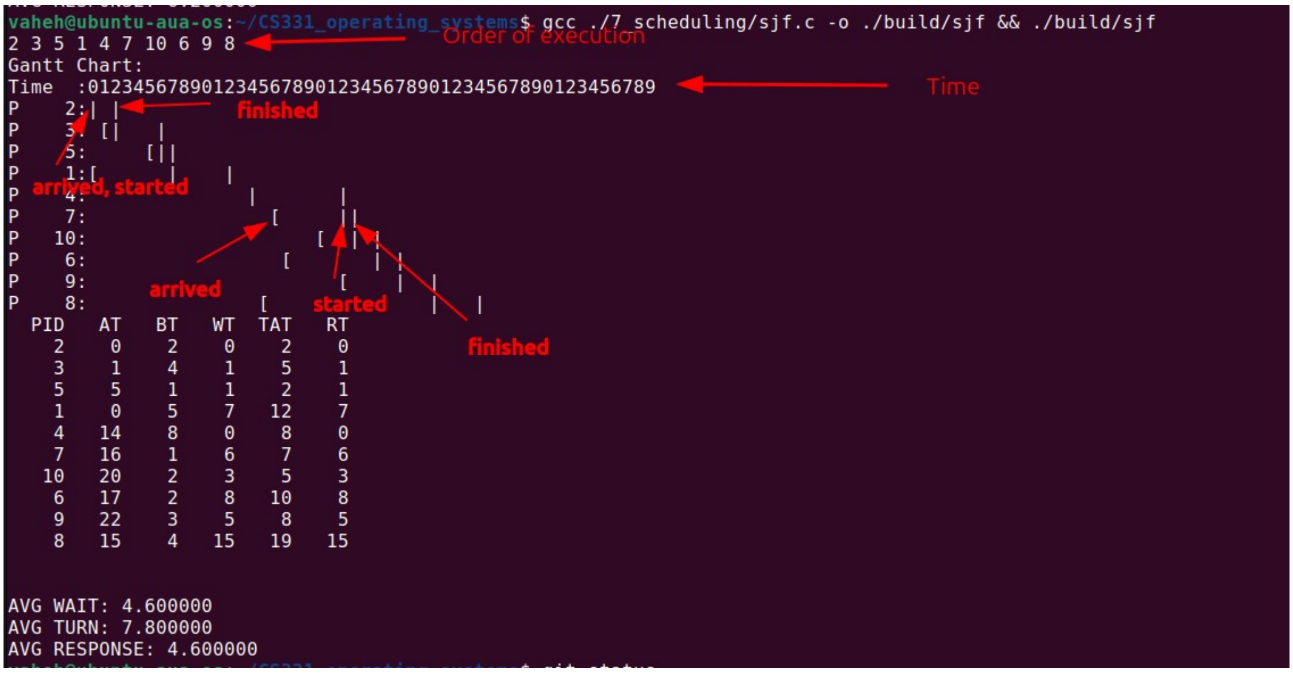
The compare function of the quicksort is also an argument, since I thought I'll just use different compare functions for the two schedulers.

At the end I free the allocated int array (don't say I need to make it null, it's at the end of the function).

Gantt Chart

At the top I print the time, so that It's more clear when each thing started. I just include one digit so that the alignment of the print doesn't mess up.

For each process, I include 2 type of progress bar. If only | | show up, it means the process arrived and started executing immediately. If [| | show up, it means that process arrived at [waited started executing at first | and ended at second |



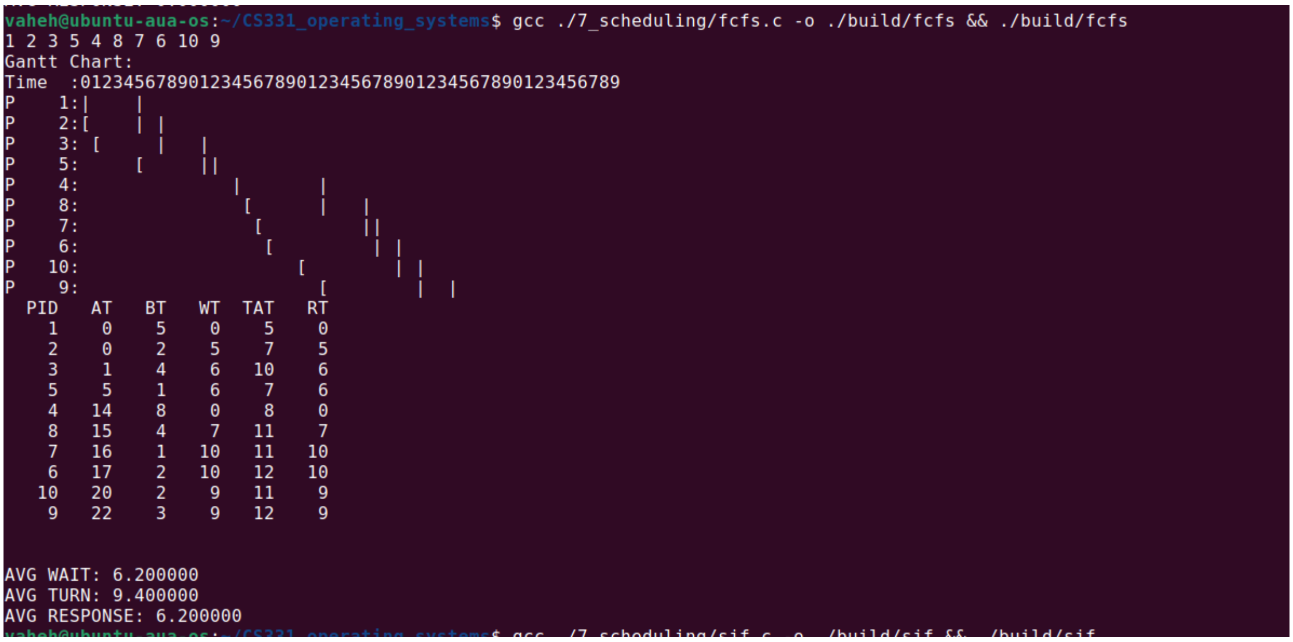
```
vaheh@ubuntu-aua-os:~/CS331_operating_systems$ gcc ./7_scheduling/sjf.c -o ./build/sjf && ./build/sjf
2 3 5 1 4 7 10 6 9 8
Gantt Chart:
Time  :01234567890123456789012345678901234567890123456789
P    2:| |
P    3: [|   |
P    5:     [||
P    1:[      |    |
P    4:              |       |
P    7:                [     ||
P   10:                    [  | |
P    6:                 [       | |
P    9:                      [    |  |
P    8:               [              |   |
  PID    AT    BT    WT   TAT    RT
    2     0     2     0     2     0
    3     1     4     1     5     1
    5     5     1     1     2     1
    1     0     5     7    12     7
    4    14     8     0     8     0
    7    16     1     6     7     6
   10    20     2     3     5     3
    6    17     2     8    10     8
    9    22     3     5     8     5
    8    15     4    15    19    15


AVG WAIT: 4.600000
AVG TURN: 7.800000
AVG RESPONSE: 4.600000
```

A VERY BIG NOTE: I did not include the functionality of expecting input from the user, because during testing I found it way more convenient to write the list directly in code, than to pass it manually every time. I could implement this, but I don't have too much time anymore. This is trivial to implement though, and I hope this won't be an issue.

fcfs.c

This just quicksorts the array by comparing arrival time.



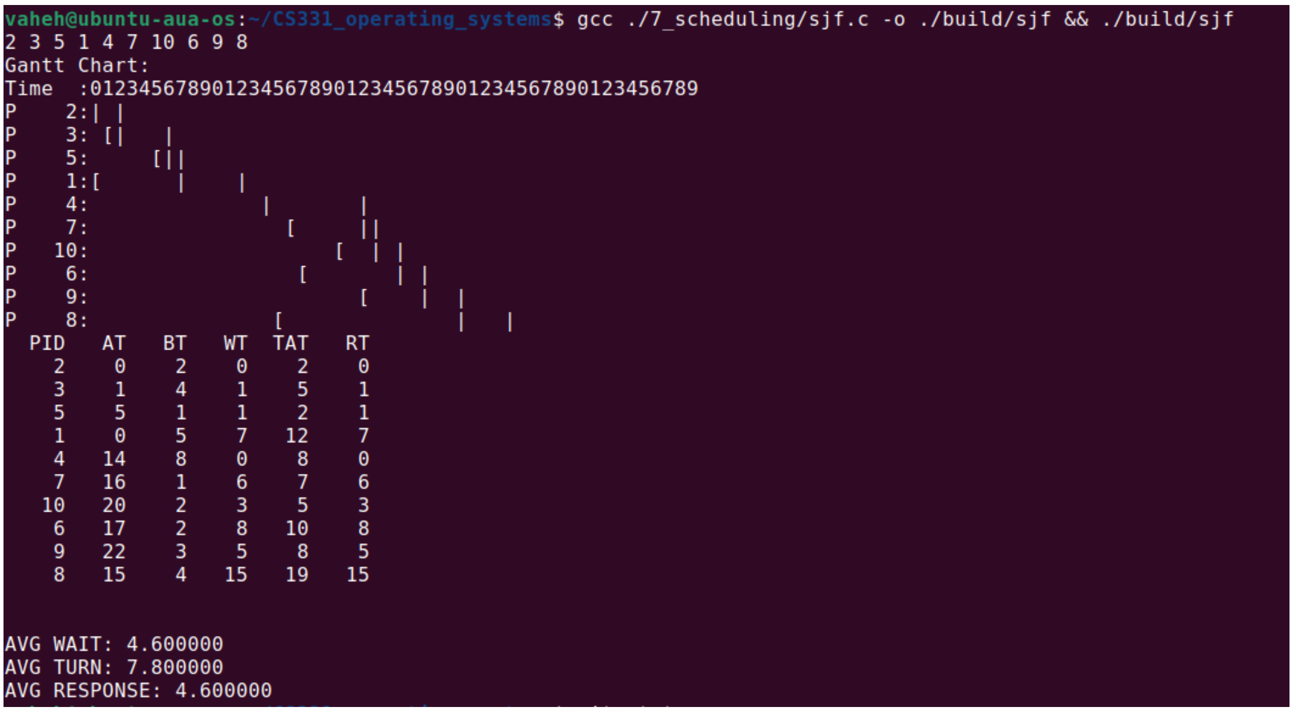
```
vaheh@ubuntu-aua-os:~/CS331_operating_systems$ gcc ./7_scheduling/fcfs.c -o ./build/fcfs && ./build/fcfs
1 2 3 5 4 8 7 6 10 9
Gantt Chart:
Time  :0123456789012345678901234567890123456789
P    1:|    |
P    2:[    | |
P    3: [     |   |
P    5:     [     ||
P    4:              |       |
P    8:               [      |   |
P    7:                [         ||
P    6:                 [         | |
P   10:                    [        | |
P    9:                      [        |  |
  PID    AT    BT    WT   TAT    RT
    1     0     5     0     5     0
    2     0     2     5     7     5
    3     1     4     6    10     6
    5     5     1     6     7     6
    4    14     8     0     8     0
    8    15     4     7    11     7
    7    16     1    10    11    10
    6    17     2    10    12    10
   10    20     2     9    11     9
    9    22     3     9    12     9


AVG WAIT: 6.200000
AVG TURN: 9.400000
AVG RESPONSE: 6.200000
vaheh@ubuntu-aua-os:~/CS331_operating_systems$ gcc ./7_scheduling/sjf.c -o ./build/sjf && ./build/sjf
```

sjf.c

Runs an insertion sort, which takes into account the latest process that's already in place. At first, it just takes the next element as the candidate, then starting from +1 from it, starts looking for a "better one" by checking only the processes that have arrived at this point (arrival_time <= time), then takes the one with the smallest burst time. If burst times are equal, chooses the one with smallest arrival time.

And for each new process that is decided to be run next, the time is updated. Either the time is + burst time if the process has arrived already, or need to wait for it to arrive, so + arrival time - time + burst time.



```
vaheh@ubuntu-aua-os:~/CS331_operating_systems$ gcc ./7_scheduling/sjf.c -o ./build/sjf && ./build/sjf
2 3 5 1 4 7 10 6 9 8
Gantt Chart:
Time  :01234567890123456789012345678901234567890123456789
P    2:| |
P    3: [|   |
P    5:     [||
P    1:[      |    |
P    4:              |       |
P    7:                [     ||
P   10:                    [  | |
P    6:                 [       | |
P    9:                      [    |  |
P    8:               [              |   |
  PID   AT   BT   WT  TAT   RT
    2    0    2    0    2    0
    3    1    4    1    5    1
    5    5    1    1    2    1
    1    0    5    7   12    7
    4   14    8    0    8    0
    7   16    1    6    7    6
   10   20    2    3    5    3
    6   17    2    8   10    8
    9   22    3    5    8    5
    8   15    4   15   19   15


AVG WAIT: 4.600000
AVG TURN: 7.800000
AVG RESPONSE: 4.600000
```

Github repo: https://github.com/VaheHayrapetyan24/CS331_operating_systems