Leonardo Curdi 11704166

Tie breaking:

- Input is read and loaded from left to right, so processes with the same arrival time will be favored in the order they are listed.
- In all 3 of my schedulers, I organized the order of the code in the cycle loop in a specific way that would respect the tie breaking rules in the documentation. The first step in the loop is to check if blocked processes are ready. In a tie situation, they must be queued up before new arrivals, as they are older. Then you can queue up new arrivals. After queueing up ready processes, you can update the current running process. It is updated in this order: check if it is finished first, then check if it is issuing an I/O request, then (in RR) check if the time quantum is up.
- Lastly, when inserting in order into a queue (blocked queue in all schedulers and ready queue in SJF), it is typical to use a < operator while iterating the queue to find the correct spot to insert. This creates a tie breaking error in our scheduler because new processes will be inserted in front of older processes with the same value. To fix this, we need to use <= so that when processes have the same value, the new one is inserted behind.

Implementation:

FCFS Alg:

```
- start a program loop
- track 2 queues, for ready processes and processes that are waiting on I/O
- at every time interval:
        - queue up all blocked processes that have just finished waiting on I/O
                - NOTE: finished blocked processes need to be queued up before new arrivals,
because they have been waiting longer (tie breaker)
        - queue up all processes that have just arrived
                - ready processes are queued up at the end of the queue
        - if there is a current running process
                - update stats of the process
                - if it is issuing an I/O: context switch it to blocked
                        - blocked processes are queued in order by I/O time remaining
                - if it is complete: toss it into the finished processes array
        - if there is no running process:
                - if there is a ready process: start it
                - else, there are no processes to run so do nothing
        - at the end of the time interval, update the stats (waiting time) for all processes that are
ready (waiting)
        - update the current time
```

RR Alg:

```
- start a program loop
- track 2 queues, for ready processes and processes that are waiting on I/O
- track a timer that counts down every quantum
- at every time interval:
        - queue up all blocked processes that have just finished waiting on I/O
```

```
            - NOTE: finished blocked processes need to be queued up before new arrivals, because they have been waiting longer (tie breaker)
        - queue up all processes that have just arrived
            - ready processes are queued up at the end of the queue
        - if there is a current running process
            - update stats of the process
            - if it is issuing an I/O: context switch it to blocked
                - blocked processes are queued in order by I/O time remaining
            - if it is complete: toss it into the finished processes array
        - if the time quantum is up:
            - if there is a process running: context switch it to ready
            - reset the quantum timer
        - if there is no running process:
            - if there is a ready process: start it
            - else, there are no processes to run so do nothing
        - at the end of the time interval, update the stats (waiting time) for all processes that are ready (waiting)
        - update the current time and the quantum timer
```

SJF Alg:

```
- start a program loop
- track 2 queues, for ready processes and processes that are waiting on I/O
- at every time interval:
    - queue up all blocked processes that have just finished waiting on I/O
        - NOTE: finished blocked processes need to be queued up before new arrivals, because they have been waiting longer (tie breaker)
    - queue up all processes that have just arrived
        - ready processes are queued up in order based on shortest total CPU time remaining
    - if there is a current running process
        - update stats of the process
        - if it is issuing an I/O: context switch it to blocked
            - blocked processes are queued in order by I/O time remaining
        - if it is complete: toss it into the finished processes array
    - if there is no running process:
        - if there is a ready process: start it
        - else, there are no processes to run so do nothing
    - at the end of the time interval, update the stats (waiting time) for all processes that are ready (waiting)
    - update the current time
```