Robert Rash
Dr. Çankaya
CSE5343
21 July 2017

Semester Project

Part 1

Processes are represented by the PCB class. These are basic class representations of actual PCBs from a real operating system. There is no real functionality in a PCB object itself aside from storing data.

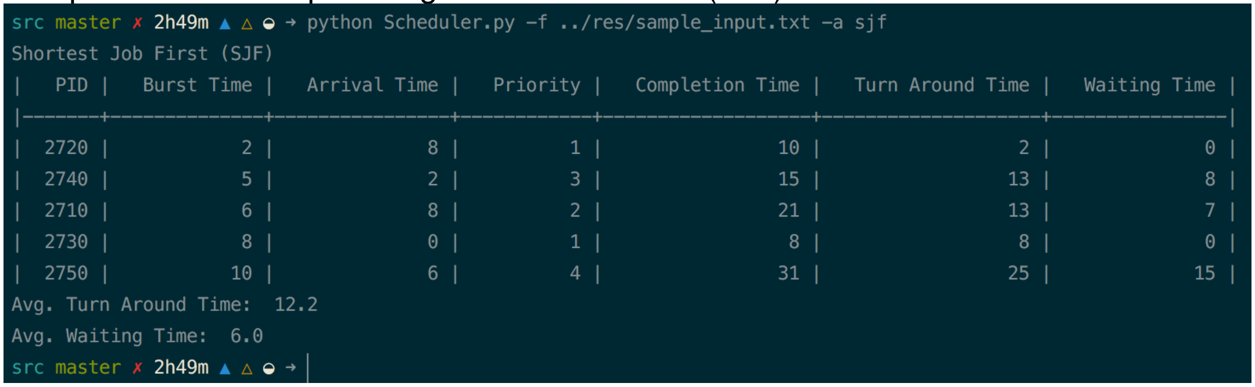
The main queue data structure is implemented using a doubly-linked list of my own creation. Basic additions and deletions can be accomplished using the push_back()/push_front() methods and the pop_back()/pop_front() methods, respectively. Additionally, there are insert() and delete(). insert() puts a value in the list at a given index location in the list, and delete() deletes a given value in a greedy fashion (that is, the first value in the list that matches the given value is deleted).

Part 2

As per the requirements for CSE5343 students, I implemented Shortest-Job-First (SJF) scheduling as well as Non-Preemptive Priority (NPP) scheduling. Source code for these algorithms can be found in the provided src/ directory.

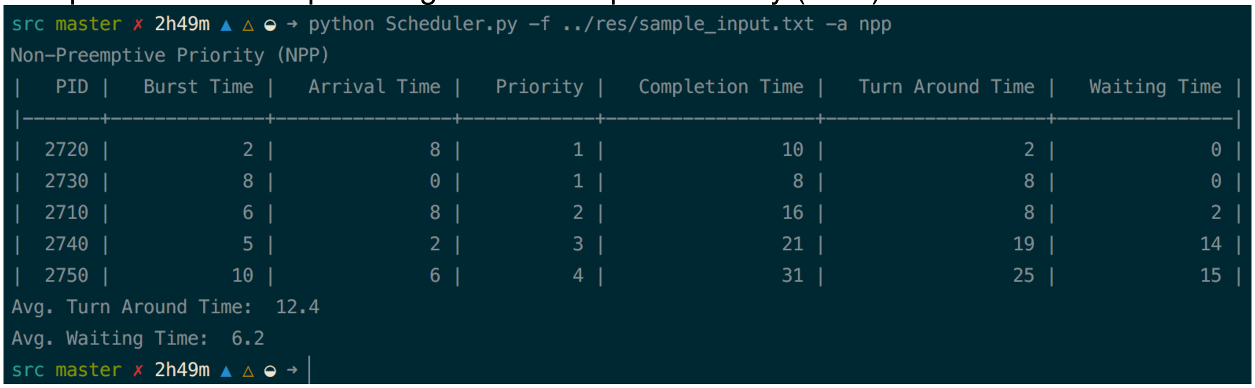
Screenshots

Sample of tabular output using Shortest Job First (SJF)



```
src master ✗ 2h49m ▲ △ ◒ ➜ python Scheduler.py -f ../res/sample_input.txt -a sjf
Shortest Job First (SJF)
|   PID |   Burst Time |   Arrival Time |   Priority |   Completion Time |   Turn Around Time |   Waiting Time |
|-------+--------------+----------------+------------+-------------------+--------------------+----------------|
|  2720 |            2 |              8 |          1 |                10 |                  2 |              0 |
|  2740 |            5 |              2 |          3 |                15 |                 13 |              8 |
|  2710 |            6 |              8 |          2 |                21 |                 13 |              7 |
|  2730 |            8 |              0 |          1 |                 8 |                  8 |              0 |
|  2750 |           10 |              6 |          4 |                31 |                 25 |             15 |
Avg. Turn Around Time:  12.2
Avg. Waiting Time:  6.0
src master ✗ 2h49m ▲ △ ◒ ➜
```

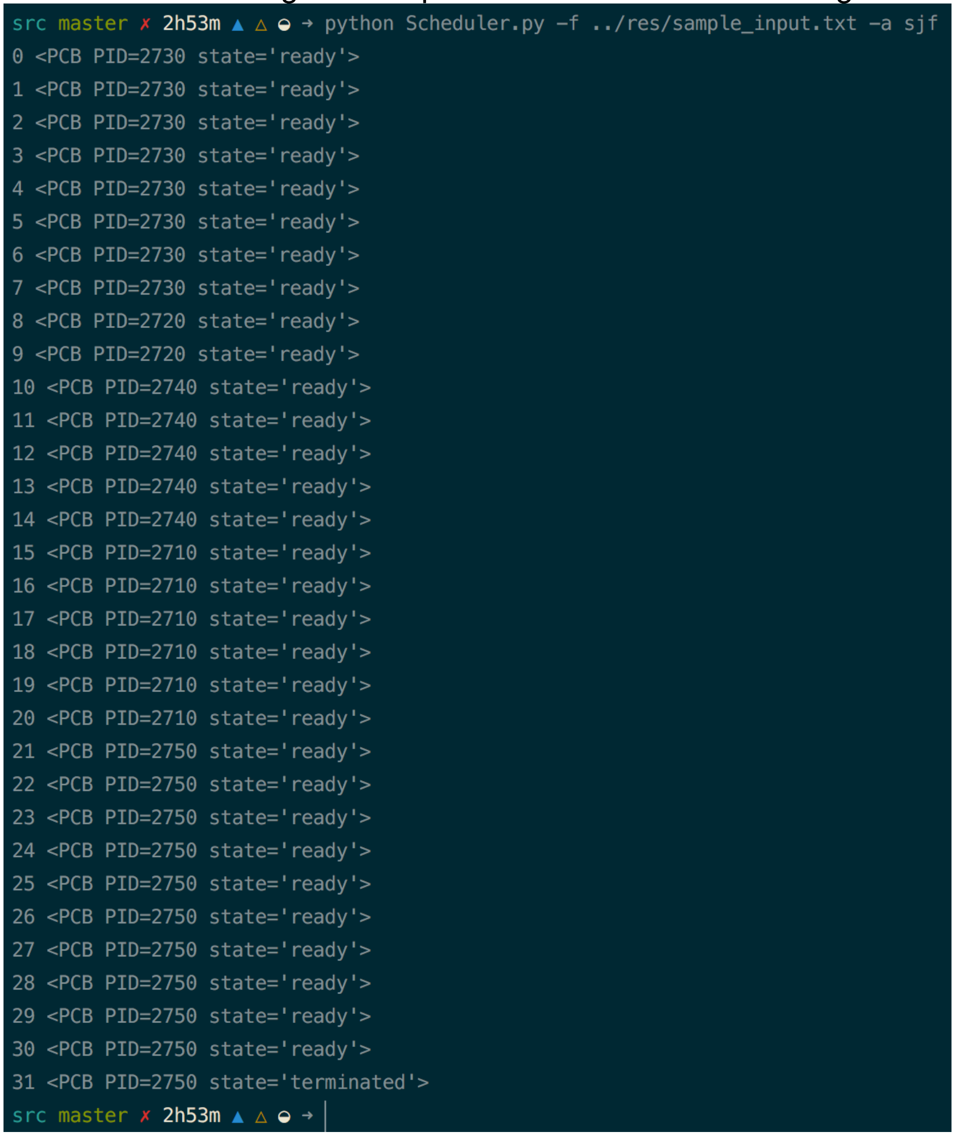
Sample of tabular output using Non-Preemptive Priority (NPP)



```
src master ✗ 2h49m ▲ △ ◒ ➜ python Scheduler.py -f ../res/sample_input.txt -a npp
Non-Preemptive Priority (NPP)
|   PID |   Burst Time |   Arrival Time |   Priority |   Completion Time |   Turn Around Time |   Waiting Time |
|-------+--------------+----------------+------------+-------------------+--------------------+----------------|
|  2720 |            2 |              8 |          1 |                10 |                  2 |              0 |
|  2730 |            8 |              0 |          1 |                 8 |                  8 |              0 |
|  2710 |            6 |              8 |          2 |                16 |                  8 |              2 |
|  2740 |            5 |              2 |          3 |                21 |                 19 |             14 |
|  2750 |           10 |              6 |          4 |                31 |                 25 |             15 |
Avg. Turn Around Time:  12.4
Avg. Waiting Time:  6.2
src master ✗ 2h49m ▲ △ ◒ ➜
```

Execution trace for Shortest Job First (SJF)

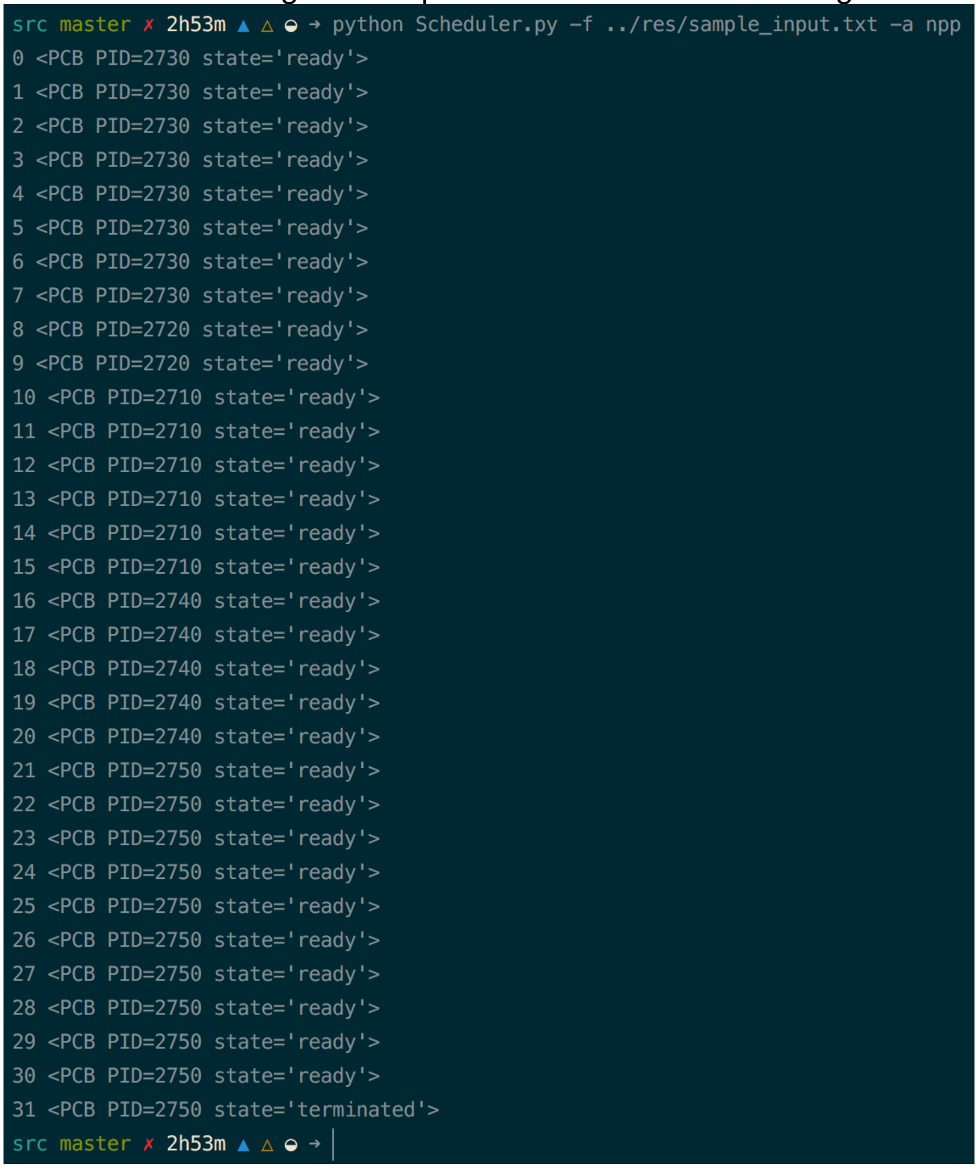
- Numbers on left indicate “system time”
- Item on right is a representation of a PCB with a given PID and state



```
src master ✗ 2h53m ▲ △ ◒ ➜ python Scheduler.py -f ../res/sample_input.txt -a sjf
0 <PCB PID=2730 state='ready'>
1 <PCB PID=2730 state='ready'>
2 <PCB PID=2730 state='ready'>
3 <PCB PID=2730 state='ready'>
4 <PCB PID=2730 state='ready'>
5 <PCB PID=2730 state='ready'>
6 <PCB PID=2730 state='ready'>
7 <PCB PID=2730 state='ready'>
8 <PCB PID=2720 state='ready'>
9 <PCB PID=2720 state='ready'>
10 <PCB PID=2740 state='ready'>
11 <PCB PID=2740 state='ready'>
12 <PCB PID=2740 state='ready'>
13 <PCB PID=2740 state='ready'>
14 <PCB PID=2740 state='ready'>
15 <PCB PID=2710 state='ready'>
16 <PCB PID=2710 state='ready'>
17 <PCB PID=2710 state='ready'>
18 <PCB PID=2710 state='ready'>
19 <PCB PID=2710 state='ready'>
20 <PCB PID=2710 state='ready'>
21 <PCB PID=2750 state='ready'>
22 <PCB PID=2750 state='ready'>
23 <PCB PID=2750 state='ready'>
24 <PCB PID=2750 state='ready'>
25 <PCB PID=2750 state='ready'>
26 <PCB PID=2750 state='ready'>
27 <PCB PID=2750 state='ready'>
28 <PCB PID=2750 state='ready'>
29 <PCB PID=2750 state='ready'>
30 <PCB PID=2750 state='ready'>
31 <PCB PID=2750 state='terminated'>
src master ✗ 2h53m ▲ △ ◒ ➜
```

Execution trace for Non-Preemptive Priority (NPP)

- Numbers on left indicate “system time”
- Item on right is a representation of a PCB with a given PID and state



```
src master ✗ 2h53m ▲ △ ◒ ➜ python Scheduler.py -f ../res/sample_input.txt -a npp
0 <PCB PID=2730 state='ready'>
1 <PCB PID=2730 state='ready'>
2 <PCB PID=2730 state='ready'>
3 <PCB PID=2730 state='ready'>
4 <PCB PID=2730 state='ready'>
5 <PCB PID=2730 state='ready'>
6 <PCB PID=2730 state='ready'>
7 <PCB PID=2730 state='ready'>
8 <PCB PID=2720 state='ready'>
9 <PCB PID=2720 state='ready'>
10 <PCB PID=2710 state='ready'>
11 <PCB PID=2710 state='ready'>
12 <PCB PID=2710 state='ready'>
13 <PCB PID=2710 state='ready'>
14 <PCB PID=2710 state='ready'>
15 <PCB PID=2710 state='ready'>
16 <PCB PID=2740 state='ready'>
17 <PCB PID=2740 state='ready'>
18 <PCB PID=2740 state='ready'>
19 <PCB PID=2740 state='ready'>
20 <PCB PID=2740 state='ready'>
21 <PCB PID=2750 state='ready'>
22 <PCB PID=2750 state='ready'>
23 <PCB PID=2750 state='ready'>
24 <PCB PID=2750 state='ready'>
25 <PCB PID=2750 state='ready'>
26 <PCB PID=2750 state='ready'>
27 <PCB PID=2750 state='ready'>
28 <PCB PID=2750 state='ready'>
29 <PCB PID=2750 state='ready'>
30 <PCB PID=2750 state='ready'>
31 <PCB PID=2750 state='terminated'>
src master ✗ 2h53m ▲ △ ◒ ➜
```

Programming Environment

My main machine is a MacBook Pro running macOS Sierra. My text editor of choice is vim.

Source code can be found attached to this project or on GitHub (request access at https://github.com/andy-rash).

This program is written in Python 2.7. I used a virtualenv in order to neatly encapsulate program requirements. In order to install the required components, install virtualenv, create a virtualenv, and run `pip install –r requirements.txt` from the root folder.