Alex Hakimzadeh
CS 3360
Dr. Kecheng Yang
29 October 2024

CS 3360 Programming Assignment 2 Report

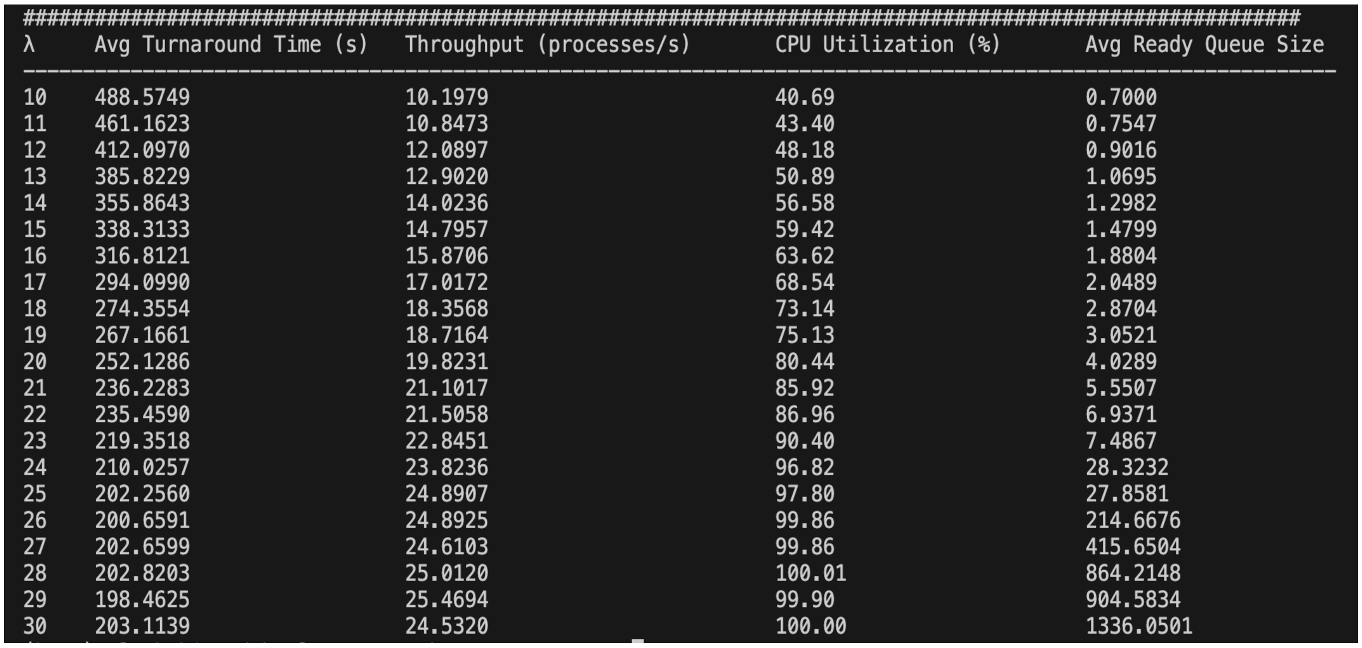
Overview

This project involved building a discrete-time event simulator to model a FCFS CPU scheduling algorithm. The primary goal was to measure the impact of varying workloads on four key performance metrics:

- Average turnaround time of processes
- Total throughput
- Average CPU utilization
- Average number of processes in the Ready Queue

The simulation assumed that processes arrived following a Poisson process and had exponentially distributed service times. The simulation was run for different arrival rates (λ) ranging from 10 to 30 processes per second, while keeping the service time fixed at an average of 0.04 seconds per process.

The simulation terminates after 10,000 processes are completed, and the metrics listed above are collected for each run.

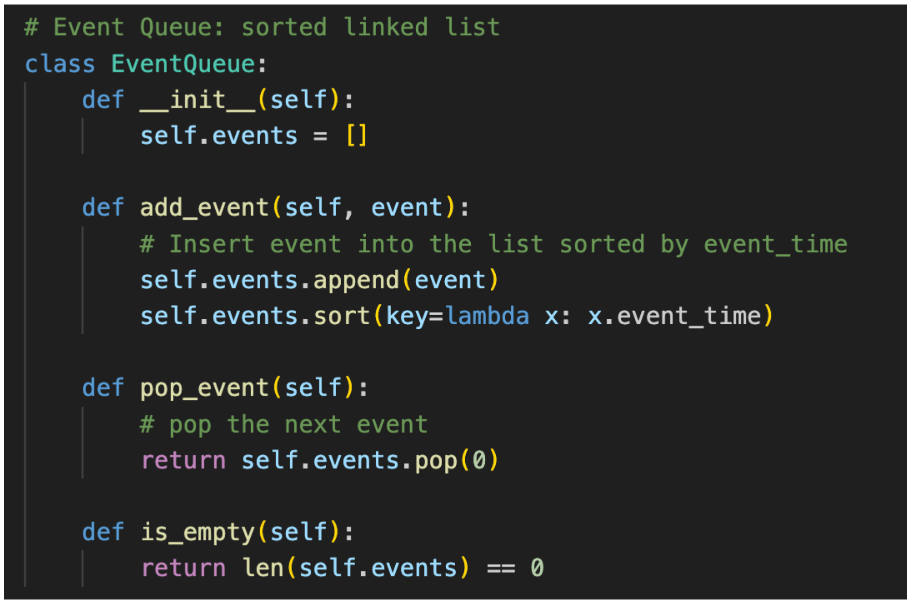


```
##################################################################################################
λ     Avg Turnaround Time (s)    Throughput (processes/s)       CPU Utilization (%)       Avg Ready Queue Size
-----------------------------------------------------------------------------------------------------------------
10    488.5749                   10.1979                        40.69                     0.7000
11    461.1623                   10.8473                        43.40                     0.7547
12    412.0970                   12.0897                        48.18                     0.9016
13    385.8229                   12.9020                        50.89                     1.0695
14    355.8643                   14.0236                        56.58                     1.2982
15    338.3133                   14.7957                        59.42                     1.4799
16    316.8121                   15.8706                        63.62                     1.8804
17    294.0990                   17.0172                        68.54                     2.0489
18    274.3554                   18.3568                        73.14                     2.8704
19    267.1661                   18.7164                        75.13                     3.0521
20    252.1286                   19.8231                        80.44                     4.0289
21    236.2283                   21.1017                        85.92                     5.5507
22    235.4590                   21.5058                        86.96                     6.9371
23    219.3518                   22.8451                        90.40                     7.4867
24    210.0257                   23.8236                        96.82                     28.3232
25    202.2560                   24.8907                        97.80                     27.8581
26    200.6591                   24.8925                        99.86                     214.6676
27    202.6599                   24.6103                        99.86                     415.6504
28    202.8203                   25.0120                        100.01                    864.2148
29    198.4625                   25.4694                        99.90                     904.5834
30    203.1139                   24.5320                        100.00                    1336.0501
```

Alex Hakimzadeh
CS 3360
Dr. Kecheng Yang
29 October 2024

System Design and Process Flow

Event Queue: A priority queue (sorted linked list) was used to handle both process arrival and departure events. The event queue ensured that events were processed in the correct chronological order.

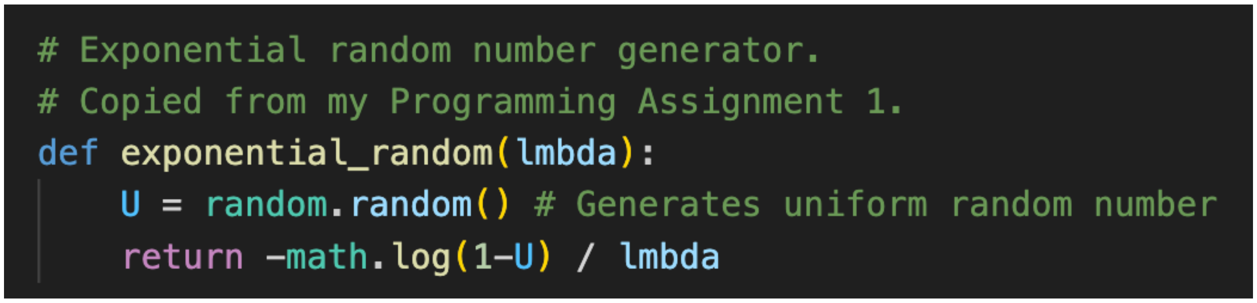


```python
# Event Queue: sorted linked list
class EventQueue:
    def __init__(self):
        self.events = []

    def add_event(self, event):
        # Insert event into the list sorted by event_time
        self.events.append(event)
        self.events.sort(key=lambda x: x.event_time)

    def pop_event(self):
        # pop the next event
        return self.events.pop(0)

    def is_empty(self):
        return len(self.events) == 0
```

- Clock: A simulation clock was used to advance time between events, updating the system state (CPU busy/idle, number of processes in the Ready Queue) at each event.

Random Number Generation

Because the use of advanced random number generators for Poisson and Exponential distributions was restricted, custom exponential random number generators were implemented. For this assignment, I copied my code on this from Assignment 1:



```python
# Exponential random number generator.
# Copied from my Programming Assignment 1.
def exponential_random(lmbda):
    U = random.random() # Generates uniform random number
    return -math.log(1-U) / lmbda
```

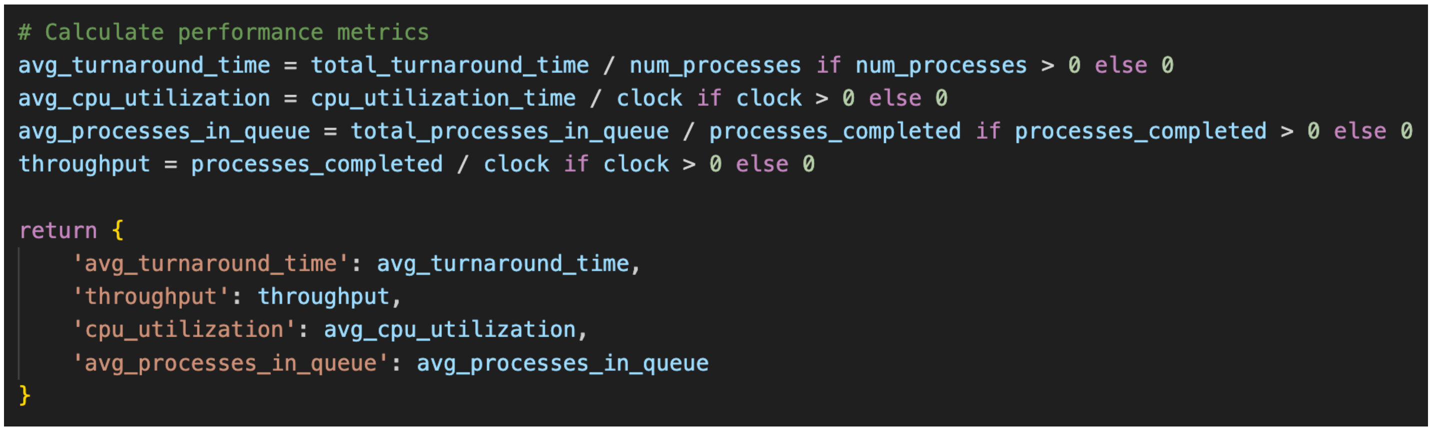
Alex Hakimzadeh
CS 3360
Dr. Kecheng Yang
29 October 2024

Performance Metrics and Simulation Runs

For each simulation run, we measured the following performance metrics:

- Average turnaround time: The average time from a process's arrival until its completion.
- Total throughput: The total number of processes completed per unit of time (processes/second).
- Average CPU utilization: The percentage of time the CPU was busy servicing processes.
- Average number of processes in the Ready Queue: The average number of processes waiting in the queue to be serviced by the CPU.

Each run simulated 10,000 processes, with the arrival rate varying between 10 and 30 processes per second in increments of 1.

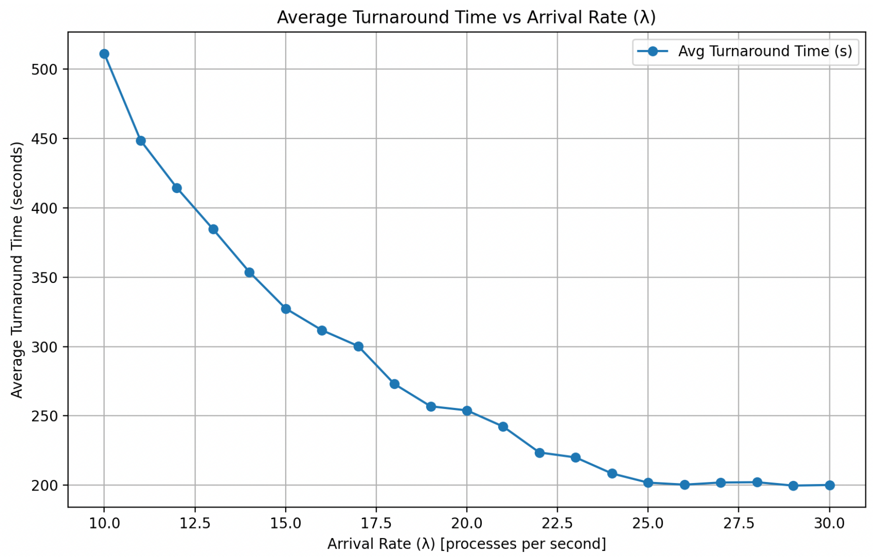


```
# Calculate performance metrics
avg_turnaround_time = total_turnaround_time / num_processes if num_processes > 0 else 0
avg_cpu_utilization = cpu_utilization_time / clock if clock > 0 else 0
avg_processes_in_queue = total_processes_in_queue / processes_completed if processes_completed > 0 else 0
throughput = processes_completed / clock if clock > 0 else 0

return {
    'avg_turnaround_time': avg_turnaround_time,
    'throughput': throughput,
    'cpu_utilization': avg_cpu_utilization,
    'avg_processes_in_queue': avg_processes_in_queue
}
```

Alex Hakimzadeh
CS 3360
Dr. Kecheng Yang
29 October 2024

Results and Interpretation Plots

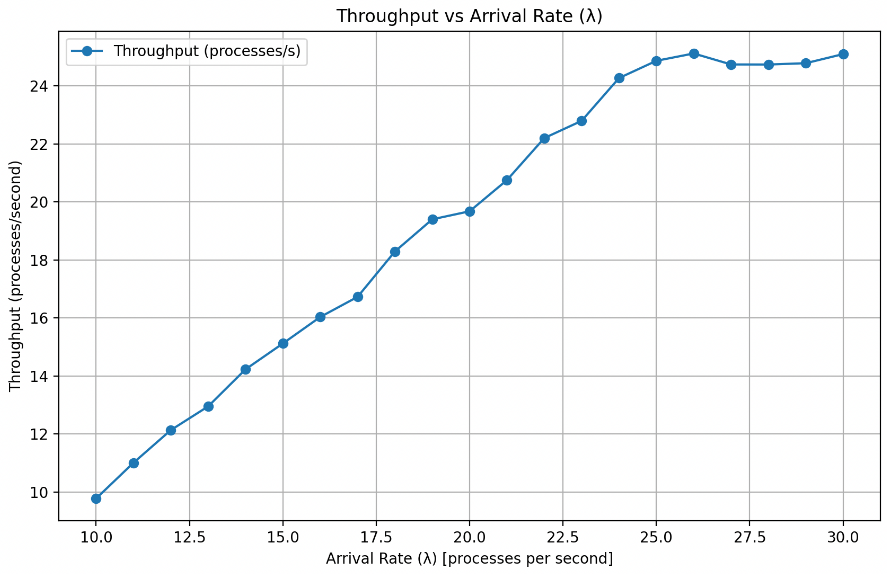
1. Average Turnaround Time vs Arrival Rate (λ)



Observation: As λ increases, the average turnaround time rises significantly until λ = around 24 then begins to level off and stabilize.

Explanation: This is expected. As arrival rates increase, initially the system is better able to handle processes, and turnaround times decrease. However, as λ increases further and approaches the CPU's maximum capacity, the CPU becomes a bottleneck, and turnaround times stabilize or increase slightly due to the fully utilized system.

2. Throughput vs Arrival Rate (λ)

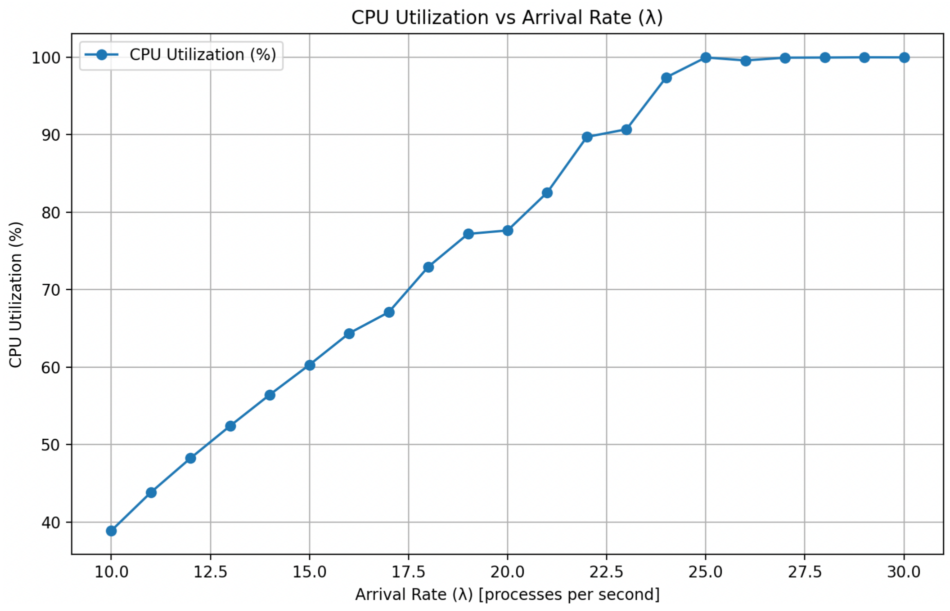


Alex Hakimzadeh
CS 3360
Dr. Kecheng Yang
29 October 2024

Observation: Throughput increases as the arrival rate increases, but it plateaus at higher values.

Explanation: The system can only complete a finite number of processes per second. Once the CPU is fully utilized, throughput stops increasing, as the CPU can handle no more than its maximum capacity.

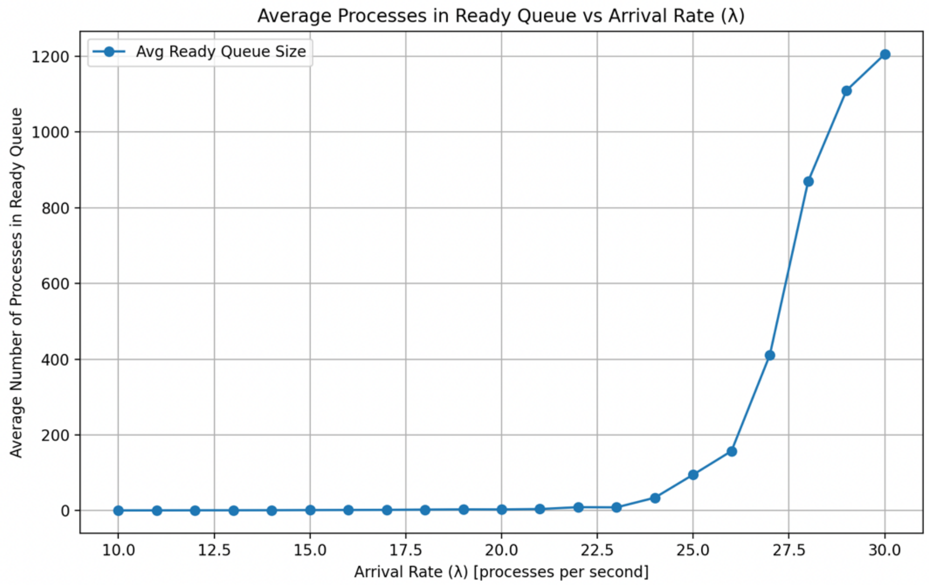
3. CPU Utilization vs Arrival Rate (λ)



Observation: CPU utilization rises steadily with and approaches 100%.

Explanation: With increasing arrival rates, the CPU spends more time servicing processes, eventually becoming fully utilized at high values of , where it is constantly busy.

4. Average Processes in the Ready Queue vs Arrival Rate (λ)



Alex Hakimzadeh
CS 3360
Dr. Kecheng Yang
29 October 2024

Observation: The average number of processes in the ready queue increases as increases.

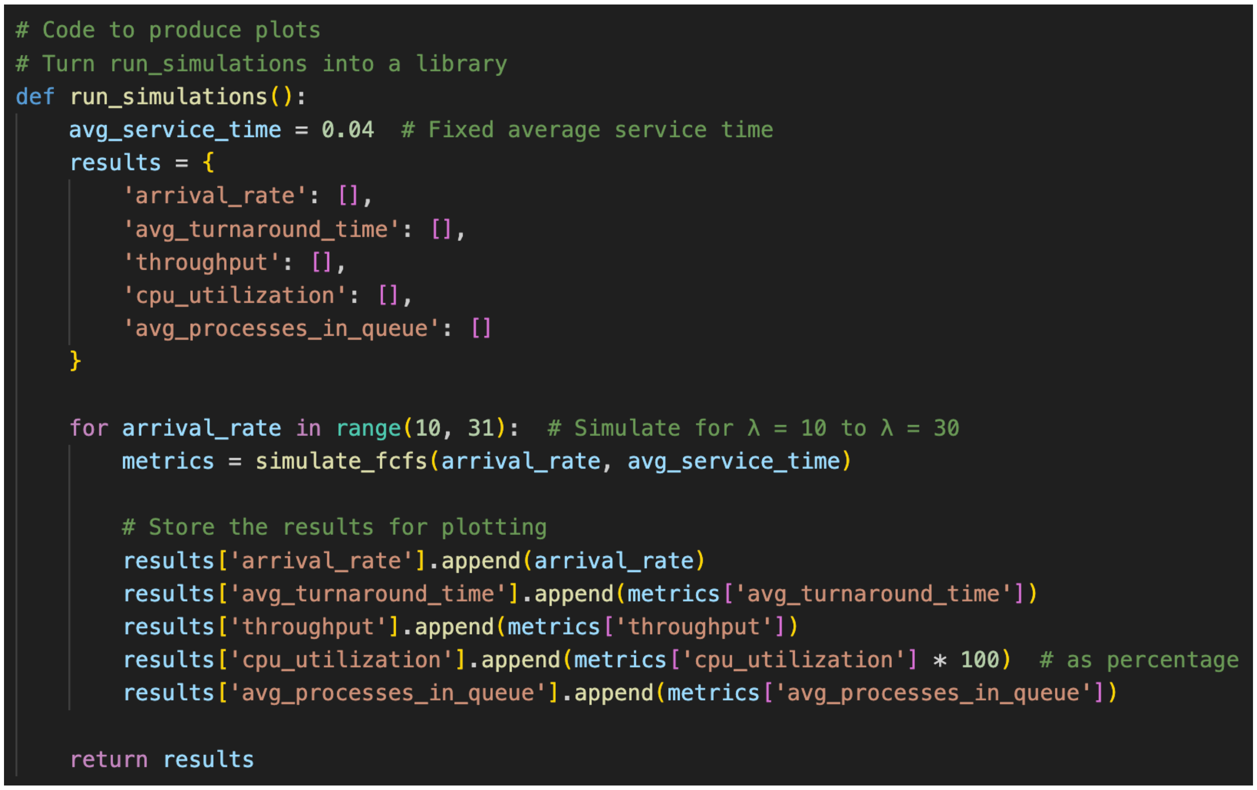
Explanation: As the arrival rate increases, more processes must wait in the queue before being serviced, leading to an increase in the average queue size.

Plot Code:

I want to use this section to highlight my code I used to automatically plot the graphs. This code is not included in my final code submission because the programming 2 assignment document specified the plots be only in the report. Yes, I realize this might've been way easier to do on pen and paper, but I needed the practice. Also, let's be honest, I couldn't possibly make the graphs as visually pleasing as the code can.

> * The original output shown on page 1 does not exactly match the output plotted on the graphs because the code for table output and graph output were captured during different runs of the program.

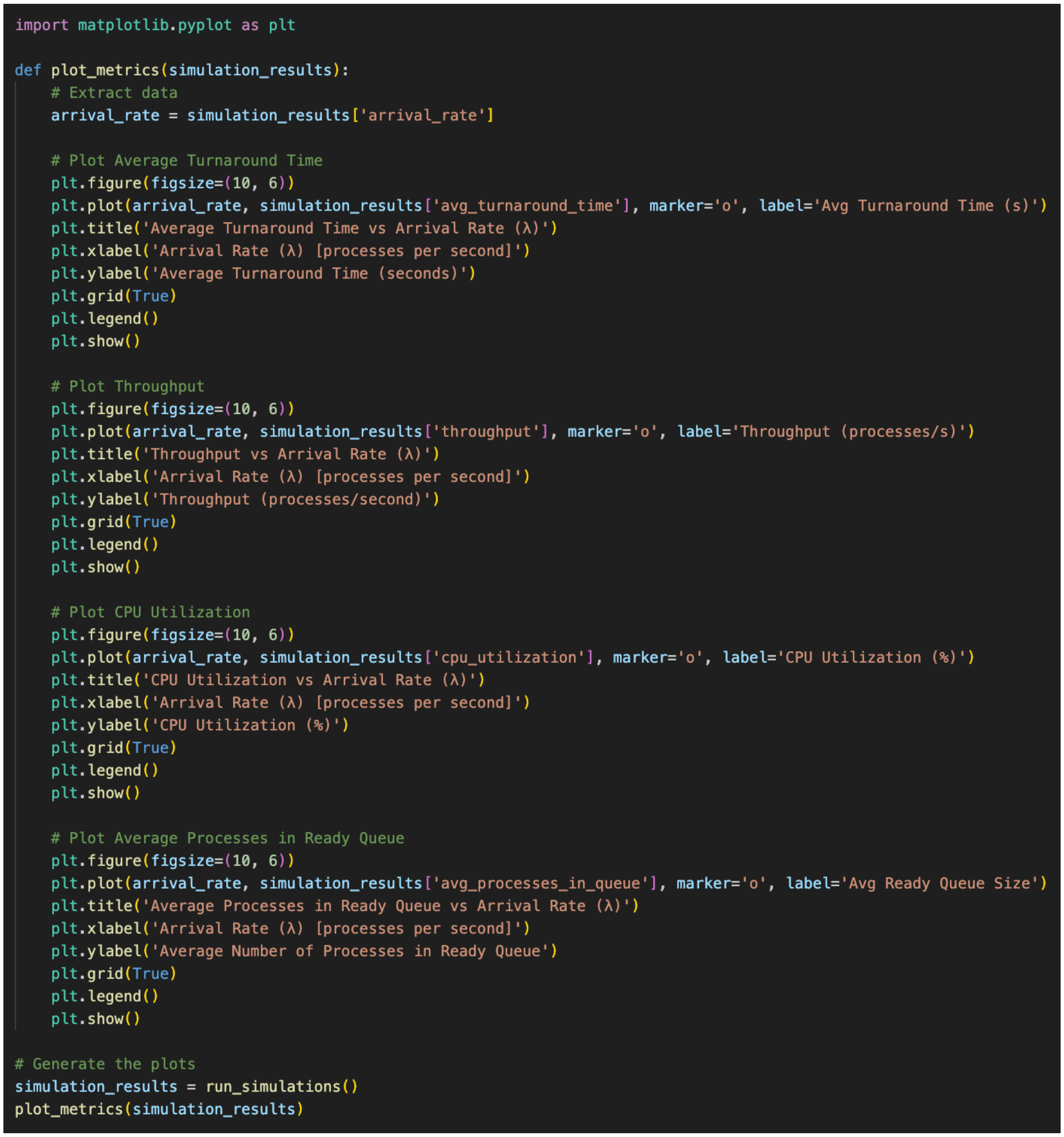
After we create the table by running the simulations we need to turn that function into a library that the "plotter" can grab from:



```python
# Code to produce plots
# Turn run_simulations into a library
def run_simulations():
    avg_service_time = 0.04  # Fixed average service time
    results = {
        'arrival_rate': [],
        'avg_turnaround_time': [],
        'throughput': [],
        'cpu_utilization': [],
        'avg_processes_in_queue': []
    }

    for arrival_rate in range(10, 31):  # Simulate for λ = 10 to λ = 30
        metrics = simulate_fcfs(arrival_rate, avg_service_time)

        # Store the results for plotting
        results['arrival_rate'].append(arrival_rate)
        results['avg_turnaround_time'].append(metrics['avg_turnaround_time'])
        results['throughput'].append(metrics['throughput'])
        results['cpu_utilization'].append(metrics['cpu_utilization'] * 100)  # as percentage
        results['avg_processes_in_queue'].append(metrics['avg_processes_in_queue'])

    return results
```

Alex Hakimzadeh
CS 3360
Dr. Kecheng Yang
29 October 2024

Then, we import matplotlib.pyplot and extract the date so we can plot the graphs. After data is extracted, we create each of the 4 graphs using that data.



```python
import matplotlib.pyplot as plt

def plot_metrics(simulation_results):
    # Extract data
    arrival_rate = simulation_results['arrival_rate']

    # Plot Average Turnaround Time
    plt.figure(figsize=(10, 6))
    plt.plot(arrival_rate, simulation_results['avg_turnaround_time'], marker='o', label='Avg Turnaround Time (s)')
    plt.title('Average Turnaround Time vs Arrival Rate (λ)')
    plt.xlabel('Arrival Rate (λ) [processes per second]')
    plt.ylabel('Average Turnaround Time (seconds)')
    plt.grid(True)
    plt.legend()
    plt.show()

    # Plot Throughput
    plt.figure(figsize=(10, 6))
    plt.plot(arrival_rate, simulation_results['throughput'], marker='o', label='Throughput (processes/s)')
    plt.title('Throughput vs Arrival Rate (λ)')
    plt.xlabel('Arrival Rate (λ) [processes per second]')
    plt.ylabel('Throughput (processes/second)')
    plt.grid(True)
    plt.legend()
    plt.show()

    # Plot CPU Utilization
    plt.figure(figsize=(10, 6))
    plt.plot(arrival_rate, simulation_results['cpu_utilization'], marker='o', label='CPU Utilization (%)')
    plt.title('CPU Utilization vs Arrival Rate (λ)')
    plt.xlabel('Arrival Rate (λ) [processes per second]')
    plt.ylabel('CPU Utilization (%)')
    plt.grid(True)
    plt.legend()
    plt.show()

    # Plot Average Processes in Ready Queue
    plt.figure(figsize=(10, 6))
    plt.plot(arrival_rate, simulation_results['avg_processes_in_queue'], marker='o', label='Avg Ready Queue Size')
    plt.title('Average Processes in Ready Queue vs Arrival Rate (λ)')
    plt.xlabel('Arrival Rate (λ) [processes per second]')
    plt.ylabel('Average Number of Processes in Ready Queue')
    plt.grid(True)
    plt.legend()
    plt.show()

# Generate the plots
simulation_results = run_simulations()
plot_metrics(simulation_results)
```