CS350 EVAL HW4 SOLUTIONS

a) Any method for checking the number of cores works. For SCC or any linux machine, we can use `lscpu` command. Also, checking through other commands/file system/UI works.

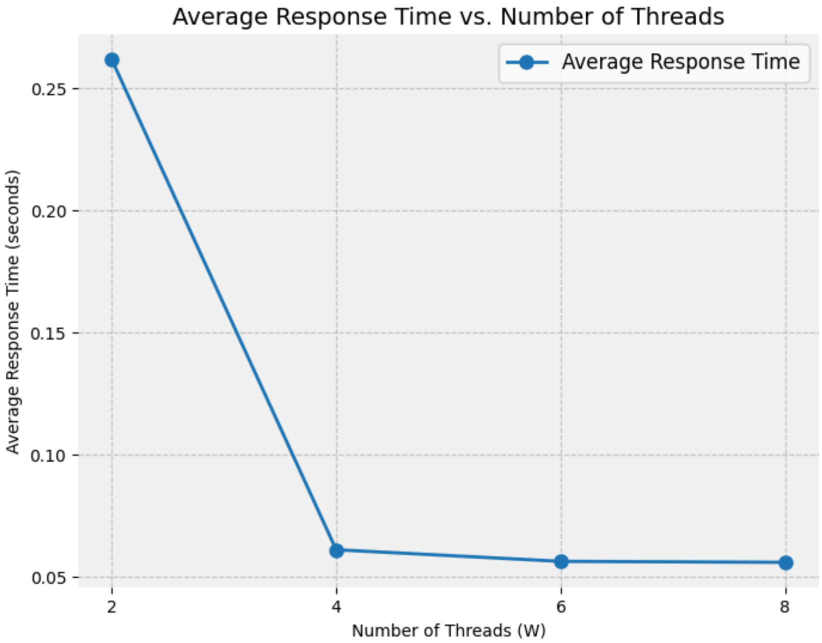
b) Calculating Utilization by each thread:

Thread T0 Utilization: 87.98%
Thread T1 Utilization: 88.26%
This makes sense since we are implementing an M/M/N system, so the workload is split by the threads equally in theory. Therefore, due to the threads picking up requests at random, the load is balanced between them.

c) Values calculated below:

W=2 Average Response Time: 0.2619942800000135 seconds
W=4 Average Response Time: 0.061134199333231665 seconds
W=6 Average Response Time: 0.056384494666786245 seconds
W=8 Average Response Time: 0.05598465133342931 seconds

Graph: (Explanation in the next page)



From the graph, we can infer that the average response time from w=2 to w=4 is supe linear, however, we notice that after that, from w=4 to w=8, the improvement in terms of average response time is negligible. This is because since our system has already reached its maximum efficiency in terms of minimum average response time, since the queue is virtually empty since four or more worker threads are constantly working on requests and processing them as soon as they can. This also implies that the waiting time is none, meaning that the response time cannot be improved further. Thereby, increasing the number of workers makes little to no difference as seen in the graph.

d) for W = 1, rejected requests =50
Rejection rate = 50/1500 = 0.03333333333333333

For W = 2, rejected requests = 0
Rejection rate = 0/1500 = 0.0

Therefore, from this statistic, we can infer that "if X is the rejection rate with 1 worker, then X/W is the rejection rate with W workers." Is NOT always true.
Although having more workers can typically lead to lower rejection rates, this decrease is not consistently proportional. If we assumed the proposed hypothesis to be true, then the rejection rate for W=2 would be 0.03333333333333333/2 = 0.01666666666 but from our data collected above, we can see that this is not the case. The hypothesis says the decrease would be linear, but the results do not agree with that. Various elements impact the rejection rate, such as request arrival rate, worker efficiency, and queue capacity. When multiple workers are employed, parallel processing can enhance the service rate, potentially reducing rejection rates. Nonetheless, the actual reduction is affected by non-linear aspects like synchronization and system overhead, necessitating system testing to establish the exact connection.

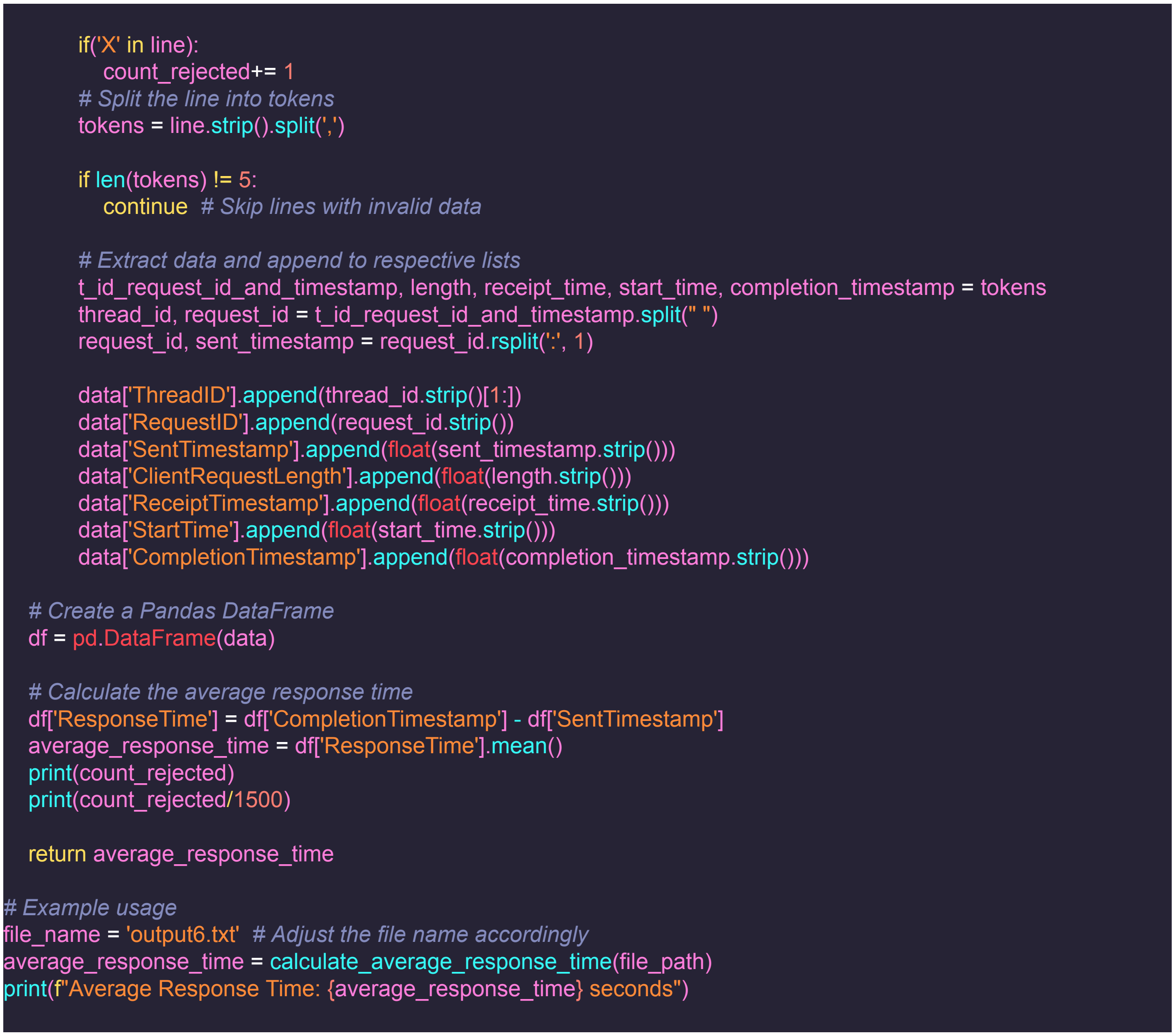
Find the code used below:

Script for Utilization calculation (part b):

```
import re

# Define a dictionary to store thread data
thread_data = {}

# Read the input file
with open("output1.txt", "r") as file:
    for line in file:
        # Use regular expressions to extract relevant data
        match = re.match(r'T(\d+) R(\d+):(\d+\.\d+),(\d+\.\d+),(\d+\.\d+),(\d+\.\d+),(\d+\.\d+)', line)
        if match:
            thread_id, request_id, sent_time, req_length, receipt_time, start_time, completion_time = match.groups()
            thread_id = int(thread_id)

            if thread_id not in thread_data:
                thread_data[thread_id] = {
                    "total_time": 0.0,
                    "busy_time": 0.0,
                }

            total_time = float(completion_time) - float(sent_time)
            busy_time = float(completion_time) - float(start_time)
            thread_data[thread_id]["total_time"] += total_time
            thread_data[thread_id]["busy_time"] += busy_time

# Calculate and print thread utilization
for thread_id, data in thread_data.items():
    total_time = data["total_time"]
    busy_time = data["busy_time"]
    utilization = (busy_time / (147199.995865-147157.896563)) * 100.0
    print(f"Thread T{thread_id} Utilization: {utilization:.2f}%")
```

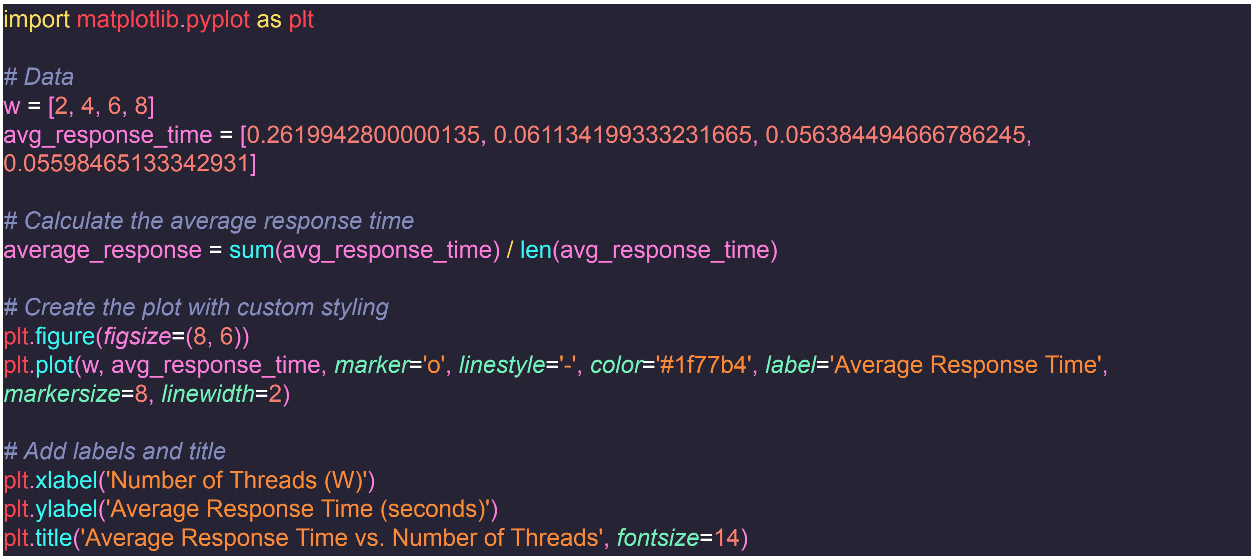
Script for Calculating avg Response time (part c):

```
import pandas as pd

def calculate_average_response_time(file_path):
    # Define the column names
    column_names = [
        'ThreadID',
        'RequestID',
        'SentTimestamp',
        'ClientRequestLength',
        'ReceiptTimestamp',
        'StartTime',
        'CompletionTimestamp'
    ]

    # Initialize empty lists to store data
    data = {col: [] for col in column_names}
    count_rejected = 0
    count_total = 0
    # Read the text file line by line
    with open(file_name, 'r') as file:
        for line in file:
            if 'Q' in line:
                continue
```

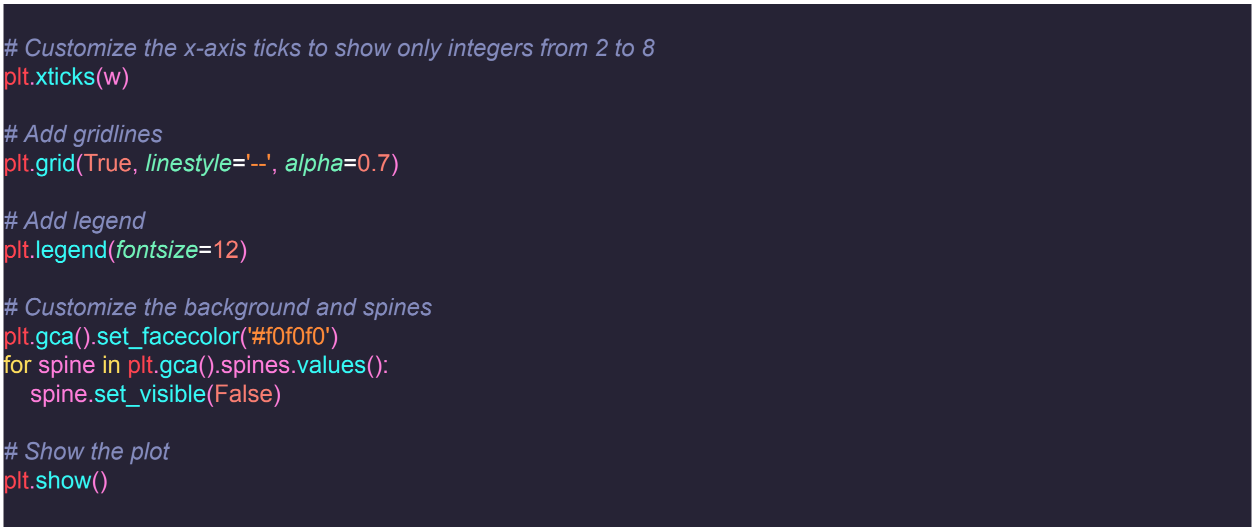


```
            if('X' in line):
                count_rejected+= 1
            # Split the line into tokens
            tokens = line.strip().split(',')

            if len(tokens) != 5:
                continue  # Skip lines with invalid data

            # Extract data and append to respective lists
            t_id_request_id_and_timestamp, length, receipt_time, start_time, completion_timestamp = tokens
            thread_id, request_id = t_id_request_id_and_timestamp.split(" ")
            request_id, sent_timestamp = request_id.rsplit(':', 1)

            data['ThreadID'].append(thread_id.strip()[1:])
            data['RequestID'].append(request_id.strip())
            data['SentTimestamp'].append(float(sent_timestamp.strip()))
            data['ClientRequestLength'].append(float(length.strip()))
            data['ReceiptTimestamp'].append(float(receipt_time.strip()))
            data['StartTime'].append(float(start_time.strip()))
            data['CompletionTimestamp'].append(float(completion_timestamp.strip()))

    # Create a Pandas DataFrame
    df = pd.DataFrame(data)

    # Calculate the average response time
    df['ResponseTime'] = df['CompletionTimestamp'] - df['SentTimestamp']
    average_response_time = df['ResponseTime'].mean()
    print(count_rejected)
    print(count_rejected/1500)

    return average_response_time

# Example usage
file_name = 'output6.txt'  # Adjust the file name accordingly
average_response_time = calculate_average_response_time(file_path)
print(f"Average Response Time: {average_response_time} seconds")
```

Script for Graph (part c as well):



```
import matplotlib.pyplot as plt

# Data
w = [2, 4, 6, 8]
avg_response_time = [0.2619942800000135, 0.061134199333231665, 0.056384494666786245,
0.05598465133342931]

# Calculate the average response time
average_response = sum(avg_response_time) / len(avg_response_time)

# Create the plot with custom styling
plt.figure(figsize=(8, 6))
plt.plot(w, avg_response_time, marker='o', linestyle='-', color='#1f77b4', label='Average Response Time',
markersize=8, linewidth=2)

# Add labels and title
plt.xlabel('Number of Threads (W)')
plt.ylabel('Average Response Time (seconds)')
plt.title('Average Response Time vs. Number of Threads', fontsize=14)
```



```
# Customize the x-axis ticks to show only integers from 2 to 8
plt.xticks(w)

# Add gridlines
plt.grid(True, linestyle='--', alpha=0.7)

# Add legend
plt.legend(fontsize=12)

# Customize the background and spines
plt.gca().set_facecolor('#f0f0f0')
for spine in plt.gca().spines.values():
    spine.set_visible(False)

# Show the plot
plt.show()
```