CS3360: Homework #6  
Due on April, 28, 2023 at 11:55PM (Firm)  
Yuvanesh Rajamani

This report is over the Discrete Time Event Simulator (FCFS) that was designed in order to assess the impact of different workloads on the following performance metrics.

* The average turnaround time of processes
* The total throughput (number of processes done per unit time)
* The CPU utilization
* The average number of processes in the Ready queue

This time the simulator will be adapted to handle 2 scenarios:

Scenario 1: Every CPU has its own Ready Queue. When a process arrives, it selects one of the CPUs uniformly at random (e.g., if we have 4 CPUs, a process would select a particular CPU with probability 0.25).  
Scenario 2: All CPUs share a global Ready Queue. When a process arrives and none of the CPUs are available, it joins the Ready Queue. Once a CPU becomes available, it would dispatch a process from the Ready Queue

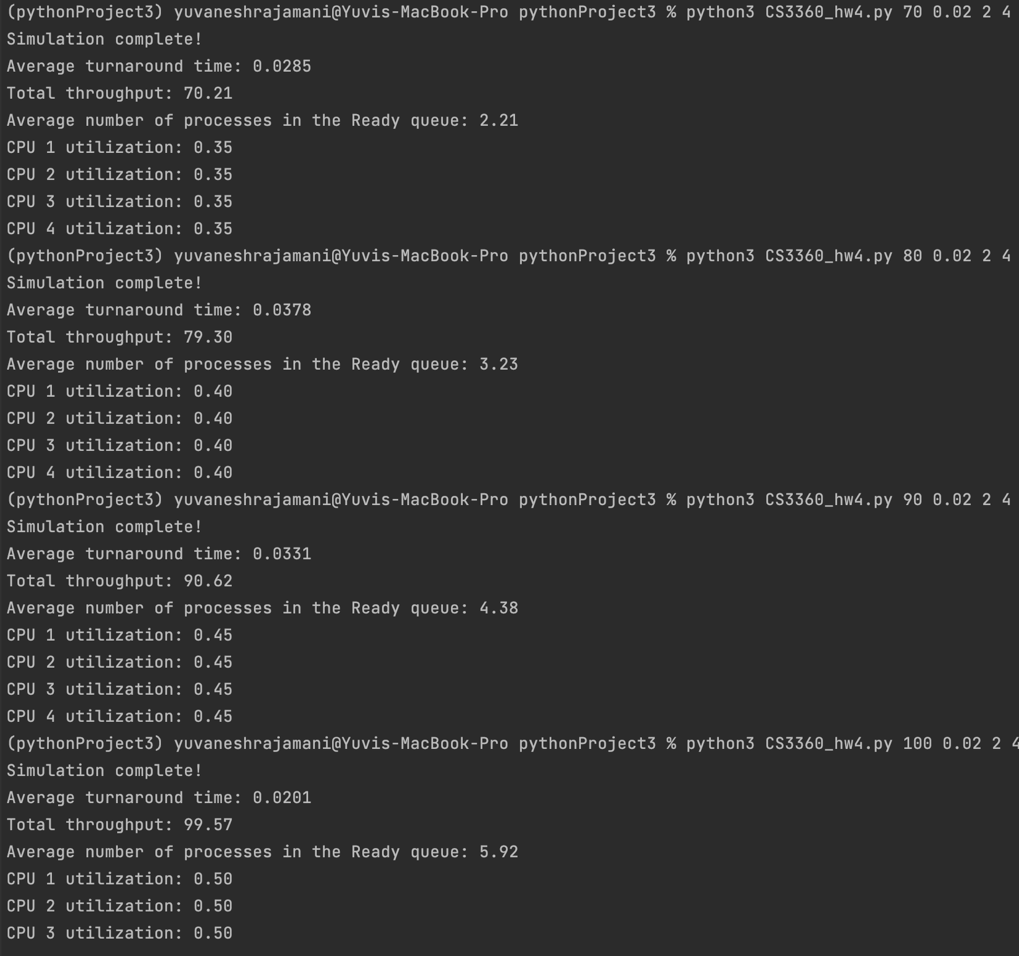
The report format is as followed

1. Intro
2. Program commands to run
3. Data with graphs
4. Analysis
5. Conclusion
6. Program Code

Program run commands:

* $ python CS3360\_hw6.py 20 0.04 1 4
* Enter the file name and the arrival rate and the average service times the scenario in which the simulation needs to run and the number of CPUs

EX program runs(four different runs:



Data: Scenario 1 Arrival rate from 50-150 with service of 0.02 and 4 CPUs

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Arrival rate | Avg turnaround time | Total Throughput | CPU 1 utilization | CPU 2 utilization | CPU 3 utilization | CPU 4 utilization | Avg #process in ready Queue |
| 50 | 0.0198 | 50.44 | 0.25 | 0.25 | 0.26 | 0.24 | 0.15 |
| 60 | 0.0502 | 59.8 | 0.29 | 0.3 | 0.29 | 0.29 | 0.22 |
| 70 | 0.0286 | 69.94 | 0.37 | 0.34 | 0.33 | 0.35 | 0.4 |
| 80 | 0.0252 | 79.49 | 0.4 | 0.39 | 0.4 | 0.42 | 0.57 |
| 90 | 0.0221 | 90.33 | 0.43 | 0.46 | 0.45 | 0.43 | 0.85 |
| 100 | 0.0397 | 100.67 | 0.5 | 0.48 | 0.5 | 0.5 | 1.21 |
| 110 | 0.0356 | 112.16 | 0.55 | 0.55 | 0.56 | 0.55 | 1.72 |
| 120 | 0.0334 | 19.83 | 0.59 | 0.61 | 0.6 | 0.62 | 2.87 |
| 130 | 0.0233 | 128.51 | 0.65 | 0.63 | 0.68 | 0.62 | 3.37 |
| 140 | 0.0284 | 140.66 | 0.71 | 0.73 | 0.68 | 0.7 | 4.87 |
| 150 | 0.027 | 148.03 | 0.76 | 0.69 | 0.76 | 0.7 | 7.12 |

Data: Scenario 2 Arrival rate from 50-150 with service of 0.02 and 4 CPUs

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Arrival rate | Avg turnaround time | Total Throughput | CPU 1 utilization | CPU 2 utilization | CPU 3 utilization | CPU 4 utilization | Avg #process in ready Queue |
| 50 | 0.021 | 49.83 | 0.25 | 0.25 | 0.25 | 0.25 | 0.91 |
| 60 | 0.0333 | 59.96 | 0.3 | 0.3 | 0.3 | 0.3 | 1.44 |
| 70 | 0.0285 | 70.21 | 0.35 | 0.35 | 0.35 | 0.35 | 2.21 |
| 80 | 0.0378 | 79.3 | 0.4 | 0.4 | 0.4 | 0.4 | 3.23 |
| 90 | 0.0331 | 90.62 | 0.45 | 0.45 | 0.45 | 0.45 | 4.38 |
| 100 | 0.0302 | 99.16 | 0.49 | 0.49 | 0.49 | 0.49 | 5.59 |
| 110 | 0.0271 | 110.33 | 0.56 | 0.56 | 0.56 | 56 | 8.54 |
| 120 | 0.0249 | 120.41 | 0.6 | 0.6 | 0.6 | 0.6 | 11.44 |
| 130 | 0.0231 | 129.74 | 0.65 | 0.65 | 0.65 | 0.65 | 16.02 |
| 140 | 0.021 | 142.55 | 0.71 | 0.71 | 0.71 | 0.71 | 22.52 |
| 150 | 0.0267 | 149.01 | 0.76 | 0.76 | 76 | 0.76 | 31.13 |

Scenario 1 Graphs:

Scenario 2 Graphs:

Analysis:

Turnaround time:

All of the data seems to make sense except for avg turnaround time. In both scenarios the Avg Turnaround time shows a weird behavior as show on the graphs. This might just be due to the randomization of the service and arrival time. This could also be due to and error on my part where im incorrectly calculation the avg turnaround time.

Total Throughput:

The data shows the throughput being around the same as the lambda with it increasing at the same rate as lambda. The data is consistent for both scenarios. This shows that the simulation is able to handle the arrival rate all the way to a lambda of 150. This makes sense considering that there are 4 CPUs being used instead of 1 so there’s no bottleneck unlike the first Discrete time event simulator we constructed.

CPU Utilization:

For scenario 1 we see that all of CPUs utilizations are increasing along with the lambda with some slight variation between the utilizations of the CPUs. This variation can be due to the fact that for scenario 1 the CPU to be used is randomly chosen. In Scenario 2 we also see the utilization for all CPUs increase along with the lambda. The only difference is that the utilization is the same for all the CPUs for a given lambda. This can be due to the fact that in scenario 2 the CPU to be assigned I based on availability.

Avg Processes in Ready Queue:

In both scenarios we see that the Average number of processes in the queue increases with the lambda. The only difference between both scenarios is that for scenario 1 we see that the number of process in the queue in much lower in all stages of lambda. This can be due to the fact the each CPU has their own ready queue making it more efficient. While in scenario 2 they all share a ready queue

Conclusion:

Overall the discrete time event simulator developed works properly expect for the data collect on some values being off. The simulator works as intended.

Program code:

import random  
import numpy as np  
import sys  
import heapq  
  
def gen\_arrivalTime():  
 return np.random.exponential(1. / float(sys.argv[1]))  
  
def gen\_departureTime():  
 return np.random.exponential(1. / (1 / float(sys.argv[2])))  
  
def select\_cpu(num\_cpus, scenario, idle, cpu\_util\_time):  
 if scenario == 1:  
 return random.randint(1, num\_cpus)  
 else:  
 min\_util = min(cpu\_util\_time)  
 return cpu\_util\_time.index(min\_util) + 1  
  
scenario = int(sys.argv[3])  
num\_cpus = int(sys.argv[4])  
  
event\_q = []  
ready\_q = []  
  
clock = 0.0  
fin\_process = 0  
generated\_processes = 0  
num\_in\_ready = 0  
idle = {i: 1 for i in range(1, num\_cpus + 1)}  
cpu\_util\_time = [0.0 for \_ in range(num\_cpus)]  
  
# Generate the first arrival event  
first\_arrival\_time = gen\_arrivalTime()  
selected\_cpu = select\_cpu(num\_cpus, scenario, idle, cpu\_util\_time)  
heapq.heappush(event\_q, (first\_arrival\_time, 1, "arr", idle[selected\_cpu], first\_arrival\_time, selected\_cpu))  
generated\_processes += 1  
  
while fin\_process <= 9999:  
 event = heapq.heappop(event\_q)  
 clock = event[0]  
 if event[2] == "arr":  
 generated\_processes += 1  
 arrival\_time = gen\_arrivalTime() + clock  
 selected\_cpu = select\_cpu(num\_cpus, scenario, idle, cpu\_util\_time)  
 heapq.heappush(event\_q, (arrival\_time, generated\_processes, "arr", idle[selected\_cpu], arrival\_time, selected\_cpu))  
 if idle[event[5]]:  
 departure\_time = gen\_departureTime() + clock  
 heapq.heappush(event\_q, (departure\_time, event[1], "dep", idle[event[5]], event[4], event[5], clock))  
 idle[event[5]] = 0  
 else:  
 ready\_q.append((event[1], "arr", idle[event[5]], event[4], event[5]))  
 elif event[2] == "dep":  
 fin\_process += 1  
 cpu\_util\_time[event[5] - 1] += event[0] - event[6]  
 if ready\_q:  
 next\_process = None  
 for i, process in enumerate(ready\_q):  
 if process[4] == event[5]:  
 next\_process = process  
 ready\_q.pop(i)  
 break  
 if next\_process:  
 departure\_time = gen\_departureTime() + clock  
 heapq.heappush(event\_q, (departure\_time, next\_process[0], "dep", idle[next\_process[4]], next\_process[3], next\_process[4], clock))  
 num\_in\_ready += len(ready\_q)  
 else:  
 idle[event[5]] = 1  
 else:  
 idle[event[5]] = 1  
  
print('Simulation complete!')  
  
# Calculate performance metrics  
average\_turnaround\_time = sum([event[4] for event in event\_q if event[2] == "dep"]) / fin\_process  
total\_throughput = fin\_process / clock  
average\_num\_in\_ready = num\_in\_ready / fin\_process  
cpu\_utilization = [util\_time / clock for util\_time in cpu\_util\_time]  
  
# Print performance metrics  
print("Average turnaround time: {:.4f}".format(average\_turnaround\_time))  
print("Total throughput: {:.2f}".format(total\_throughput))  
print("Average number of processes in the Ready queue: {:.2f}".format(average\_num\_in\_ready))  
for i, util in enumerate(cpu\_utilization):  
 print("CPU {} utilization: {:.2f}".format(i + 1, util))