Name: abhishek michael

Roll No: 01

Date: 05/03/24

Practical No:18

Q10.1

import numpy as np

# Arrival rate (customers per hour)

arrival\_rate = 15

# Service rate (customers per hour)

service\_rate = 60 / 3 # Convert service rate to customers per hour

# Utilization of the teller

utilization = arrival\_rate / service\_rate

# Average number in line (L)

L = arrival\_rate / (service\_rate - arrival\_rate)

# Average number in the system (Ls)

Ls = L + utilization

# Average waiting time in line (Wq)

Wq = L / arrival\_rate

# Average waiting time in system (Ws)

Ws = Wq + 1 / service\_rate

print("Utilization of the teller:", utilization)

print("Average number in line:", L)

print("Average number in system:", Ls)

print("Average waiting time in line:", Wq, "hours")

print("Average waiting time in system:", Ws, "hours")

Utilization of the teller: 0.75

Average number in line: 3.0

Average number in system: 3.75

Average waiting time in line: 0.2 hours

Average waiting time in system: 0.25 hours

Q10.2

arrival\_rate = 15

service\_rate = 20

num\_cars = 3

# Calculate Pn for n from 0 to num\_cars

probabilities = [(1 - arrival\_rate / service\_rate) \* (arrival\_rate / service\_rate) \*\* n for n in range(num\_cars + 1)]

# Calculate the present level of service

present\_service\_level = sum(probabilities)

print("Present level of service for three or fewer cars:", present\_service\_level)

Present level of service for three or fewer cars: 0.68359375

In [ ]:



arrival\_rate **=** 15

Q10.3

import numpy as np

import simpy

class PartsDepartment:

def \_\_init\_\_(self, env, clerk\_rate):

self.env = env

self.parts\_clerk = simpy.Resource(env, capacity=1)

self.clerk\_rate = clerk\_rate

def fill\_request(self):

yield self.env.timeout(np.random.exponential(1/self.clerk\_rate))

def mechanic(env, name, parts\_dept):

arrival\_time = env.now

print(f"{name} arrives at time {arrival\_time:.2f}")

with parts\_dept.parts\_clerk.request() as request:

yield request

print(f"{name} gets served at time {env.now:.2f}")

yield env.process(parts\_dept.fill\_request())

print(f"{name} is done at time {env.now:.2f}")

def generate\_mechanics(env, parts\_dept, arrival\_rate):

while True:

yield env.timeout(np.random.poisson(1/arrival\_rate))

env.process(mechanic(env, f'Mechanic {np.random.randint(1, 100)}', parts\_dept))

def main():

clerk\_rate = 20 # parts clerk fill rate per hour

arrival\_rate = 40 # mechanics arrival rate per hour

env = simpy.Environment()

parts\_dept = PartsDepartment(env, clerk\_rate)

env.process(generate\_mechanics(env, parts\_dept, arrival\_rate))

env.run(until=100) # Simulation time in hours

if \_\_name\_\_ == '\_\_main\_\_':

main()

Mechanic 25 is done at time 99.75

Mechanic 7 gets served at time 99.75

Mechanic 7 is done at time 99.79

Mechanic 71 gets served at time 99.79

Mechanic 71 is done at time 99.85

Mechanic 77 gets served at time 99.85

Mechanic 77 is done at time 99.88

Mechanic 25 gets served at time 99.88

Mechanic 25 is done at time 99.96

Mechanic 53 gets served at time 99.96

Q10.4

import numpy as np

def calculate\_cost(N, T, S, U, machine\_idle\_cost, repairer\_cost):

# Efficiency factor

F = T / (T + U)

# Average number of machines waiting

L = N \* (1 - F)

# Average number of machines being serviced

H = N \* F \* S

# Number of machines down

machines\_down = L + H

# Cost for machines being down

total\_machine\_cost = machines\_down \* machine\_idle\_cost

# Total cost including labor

if S == 1:

total\_cost = total\_machine\_cost + repairer\_cost

else:

total\_cost = total\_machine\_cost + (repairer\_cost \* S)

return total\_cost

def main():

N = 4 # Number of machines

T = 7.5 / 60 # Time required to service a machine in hours

U = 1 # Average time a machine runs before requiring service in hours

machine\_idle\_cost = 40 # Machine idle cost per hour

repairer\_cost = 7 # Cost for hiring a repairer per hour

# Cost with one repairer

cost\_one\_repairer = calculate\_cost(N, T, 1, U, machine\_idle\_cost, repairer\_cost)

print("Cost with one repairer:", cost\_one\_repairer)

# Cost with two repairers

cost\_two\_repairers = calculate\_cost(N, T, 2, U, machine\_idle\_cost, repairer\_cost)

print("Cost with two repairers:", cost\_two\_repairers)

# Compare costs

if cost\_one\_repairer < cost\_two\_repairers:

print("Hiring one repairer is the better choice.")

elif cost\_one\_repairer > cost\_two\_repairers:

print("Hiring two repairers is the better choice.")

else:

print("Both options have the same cost.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

Cost with one repairer: 167.0

Cost with two repairers: 191.77777777777777

Hiring one repairer is the better choice.

Q10.5

import math

def calculate\_waiting\_time(N, xa, s, Sa, Ss):

# Step 1: Calculate expected customer arrival rate (λ), service rate per server (u),

# and coefficient of variation for the interarrival time (Ca) and service time (Cs)

λ = 1 / xa

u = 1 / s

Ca = Sa \* xa

Cs = Ss \* s

# Step 2: Calculate the expected server utilization (p)

p = λ / N

# Step 3: Calculate the expected number of people waiting (Lq) and the length of the wait (Wq)

Lq = (p \*\* 2 \* (2 \* (N + 1))) / (2 \* (1 - p))

Lq \*= (Ca \*\* 2 + Cs \*\* 2) / 2

Wq = Lq / λ

return Wq

def main():

N = 9 # Number of operators

xa = 0.5 # Average time between call arrivals in minutes

s = 4 # Average time to service a call in minutes

Sa = 0.203 # Standard deviation of interarrival time in minutes

Ss = 2.5 # Standard deviation of service time in minutes

# Calculate waiting time with 9 operators

waiting\_time\_9\_operators = calculate\_waiting\_time(N, xa, s, Sa, Ss)

print("Waiting time with 9 operators:", waiting\_time\_9\_operators, "minutes")

# Calculate waiting time with 10 operators

N = 10

waiting\_time\_10\_operators = calculate\_waiting\_time(N, xa, s, Sa, Ss)

print("Waiting time with 10 operators:", waiting\_time\_10\_operators, "minutes")

# Calculate waiting time with 11 operators

N = 11

waiting\_time\_11\_operators = calculate\_waiting\_time(N, xa, s, Sa, Ss)

print("Waiting time with 11 operators:", waiting\_time\_11\_operators, "minutes")

if \_\_name\_\_ == "\_\_main\_\_":

main()

Waiting time with 9 operators: 15.87465115079365 minutes

Waiting time with 10 operators: 13.751416559375 minutes

Waiting time with 11 operators: 12.12246087878788 minutes

In [ ]:



​

​