STUDY OF A MANUFACTURING FACILITY
SYSC 5001
Project Final Report

Advisor:
Prof. Chang Cheng Huang
Dept. of Systems and Computer Engineering
Carleton University

Group 101:

101206784 Abdul Mutakabbir

STUDY OF A MANUFACTURING FACILITY
SYSC 5001
Project Final Report

Table of Contents

Problem Formulation: 4
- Additional requirements include the following: 4
- Exact Problem: 4

The setting of Objectives and Overall Project Plan: 5
- Metrics for evaluation: 5
- Schedule: 5
- Resources: 5

Model Conceptualization: 6
- Conceptual Model: 6
- Data Collected: 7

Model Translation: 8
- Programming Language: 8
- Model Implementation 8
  - Code Execution: 8
- UML Diagrams: 9

Identify Distribution (Histogram): 10

Evaluate Distributions (Q-Q Plots): 11

Goodness of Fit (Chi-Square Test): 13
- Workstation 1 Processing Time: 13
- Workstation 2 Processing Time: 14
- Workstation 3 Processing Time: 15
- Inspector 1 Service Time for Component 1: 16
- Inspector 2 Service Time for Component 2: 17
- Inspector 2 Service Time for Component 2: 18

Generate Input Based on Model: 19
- Procedure: 19
- Resultant Distribution: 20
- Frequency Test (K-S Test): 20
- Test for Autocorrelation: 22
- Code: 24

Model Verification and Validation: 25
- Parameters of Interest: 25

Peer Review: 25
Little's Law: 25
Operational Model: 27
Trial Run: 28
Validation: 29
Production Runs and Analysis: 29
Production Run 1: 29
Production Run 2: 30
Production Run 3: 30
Production Run 4: 31
Production Run 5: 32
Production Run 6: 32
Notes: 33
Initialization Phase: 33
Confidence Interval: 33
Alternative Operation Policy: 35
Alternative Conceptual Model: 35
Alternative Design Flow Chart: 37
Alternative Design Policy Production Run: 38
Comparison: 39
Consequences of Alternative Policy: 39
Conclusion: 40

STUDY OF A MANUFACTURING FACILITY
SYSC 5001
Project Final Report

Problem Formulation:

Conduct a complete simulation study of this manufacturing facility. You may use any programming language to implement your simulation model.

Additional requirements include the following:

- Statistical justification/validation of the random aspects of the model (input modeling)
- Steady-state estimates of the quantities of interest accompanied by 95% confidence intervals with a width that does not exceed 20% of the estimated values
- At least one recommendation for an alternative operating policy in the facility

Exact Problem:

Conduct a simulation study for the manufacturing facility focusing on the following:

- Facility Throughput (product output per unit time)
- Probability of workstation being busy
- Average buffer occupancy
- Inspector block time

Once the simulation is conducted suggest alternative policies that can be used to better the performance based on statistical data comparison.

The setting of Objectives and Overall Project Plan:

A simulation study is to be conducted to assess the performance of this manufacturing facility, partly based on observed historical data of the inspectors' and workstations' service times.

An additional objective is to possibly improve the policy that Inspector 1 follows when delivering C1 components to the different workstations, to increase throughput and/or decrease the inspector “blocked" time.

Metrics for evaluation:

- Facility Throughput (product output per unit time)
- Probability of workstation being busy
- Average buffer occupancy
- Inspector block time

Schedule:

The project will be delivered in Four phases. The dates are as follows:

- Phase 1 – 10th February 2022
- Phase 2 – 11th March 2022
- Phase 3 – 29th March 2022
- Phase 4 – 12th April 2022

Phase 1 will provide the initial documentation, soft code, and the system architecture. Phase 2 will provide a statical implementation of the simulation. Phase 3 will include production runs, analytics, modal verification, and modal validation. Finally, Phase 4 will provide an alternate policy, conclusion, and final report.

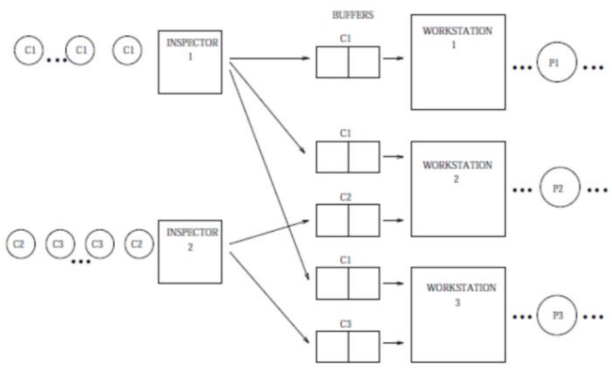
Resources:

S. No.	Position	Hourly Rate	Expected Hours	Total Cost Estimate
1	Developer	$15/hr	40hrs	$600
2	Tester	$15/hr	10hrs	$150
Total:				$750

Model Conceptualization:

Conceptual Model:

A manufacturing facility assembles three different types of products (P1, P2, P3) and has different components (C1, C2, C3) as follows:



- P1: C1
- P2: C1, C2
- P3: C1, C3

Two inspectors (I1, I2) clean and repair the components as follows:

- I1: C1
- I2: C2, C3 (Randomly)

The inspectors will never have to wait for components. There is an infinite inventory of them always immediately available.

There are three workstations in the facility, named W1, W2, and W3, which assemble products P1, P2, and P3, respectively. After the components pass inspection, they are sent to their respective workstations. Each workstation has a buffer capacity of two components, with one buffer available for each of the component types needed. A product can begin being assembled only when components of all types required are available. If all workstation buffers for a specific type of component are full, the corresponding inspector who finished inspecting a component with the same type is considered “blocked" until there is an opening, at which time the inspector can resume processing and sending components of that type.

In the present mode of operation, Inspector 1 routes components C1 to the buffer with the smallest number of components in waiting (i.e., a routing policy according to the shortest queue). In the case of a tie, W1 has the highest and W3 has the lowest priority.

Data Collected:

Historical data of the inspectors' and workstations' service times given in units of minutes as in the following files

- Inspector 1 inspection time: servinsp1.dat
- Inspector 2 inspection time for component 2: servinsp22.dat
- Inspector 2 inspection time for component 3: servinsp23.dat
- Workstation 1 processing time: ws1.dat
- Workstation 2 processing time: ws2.dat
- Workstation 3 processing time: ws3.dat

Model Translation:

Programming Language:

Python was chosen as the programming language. The reason for choosing Python is that it has all the statistical tools and functions prebuilt into it. Additionally, there is no need to learn it as it is not a specialized tool. This comes in hand with the short project deliverables time.

Model Implementation

The code for this project is divided into 7 classes as follows:

- Simulator
- FEL
- Component
- Inspector
- Buffer
- Workstation
- Product

They perform functions based on what the name suggests. The main logic of the simulation is present in the Simulator Class file (src/Simulator.py). A detailed description of the code is available in the files themselves as comments. A high-level view of the architecture is available in the UML Diagram section.

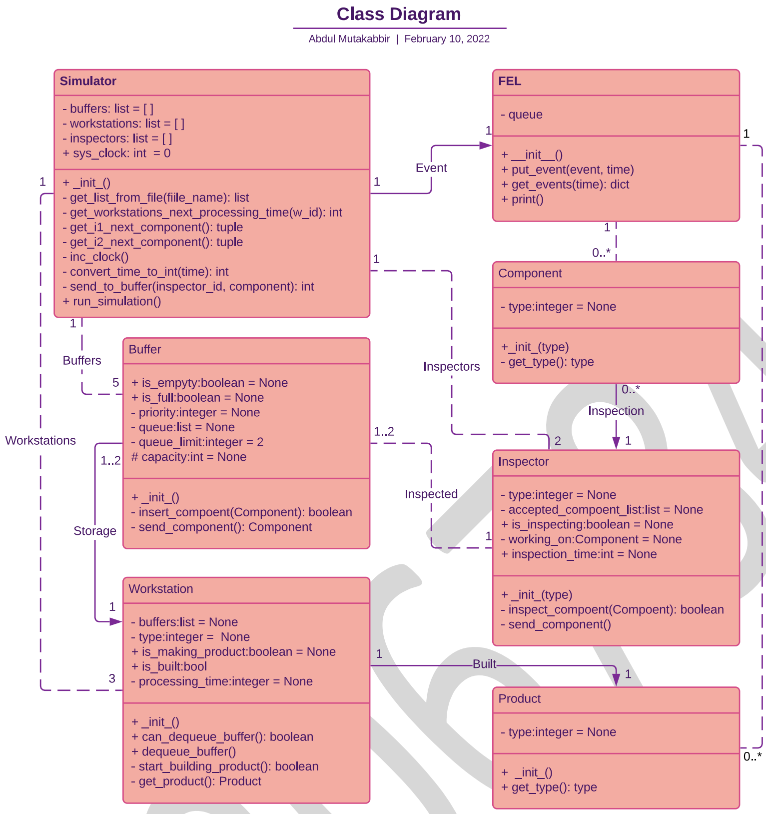
Code Execution:

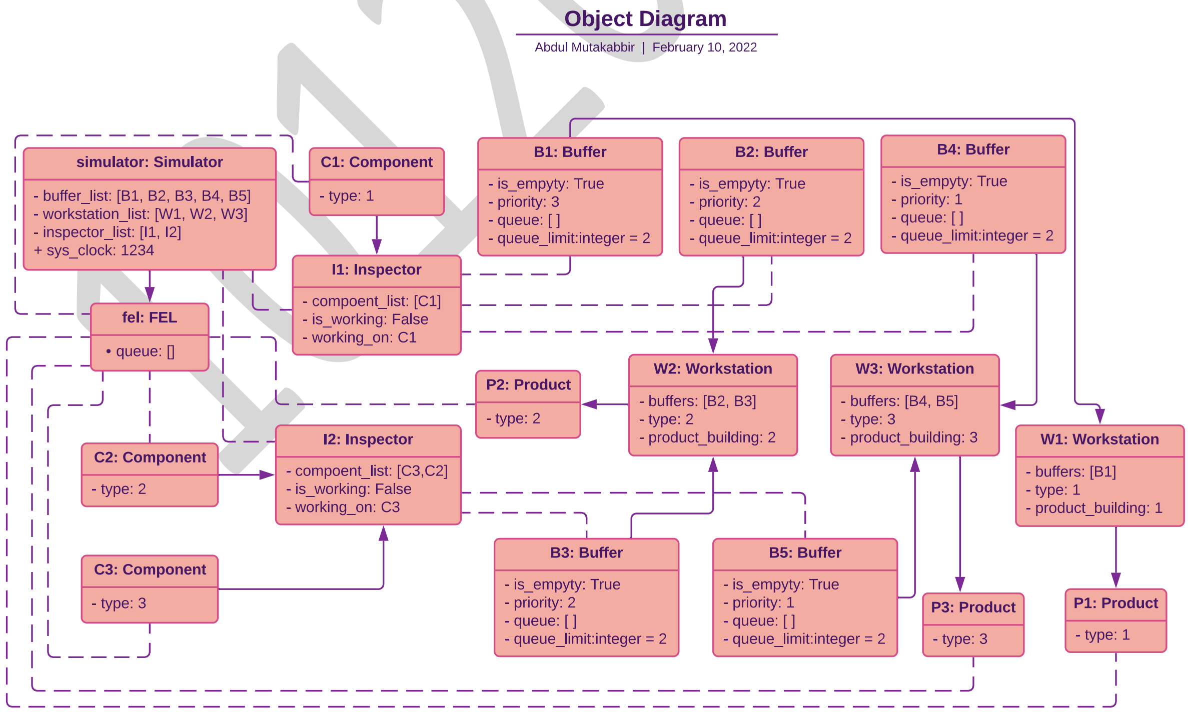
The following steps need to be followed to execute the code:

- Go to project directory
- Install all the modules from "./requirements.txt"
- Run "python src/main.py"

STUDY OF A MANUFACTURING FACILITY
SYSC 5001
Project Final Report

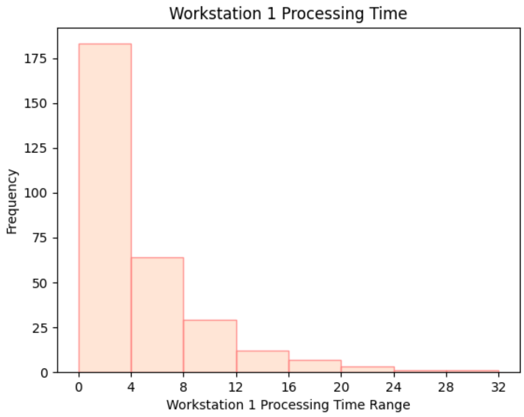
UML Diagrams:

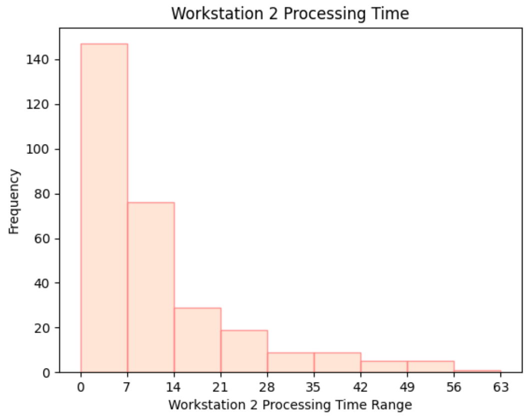


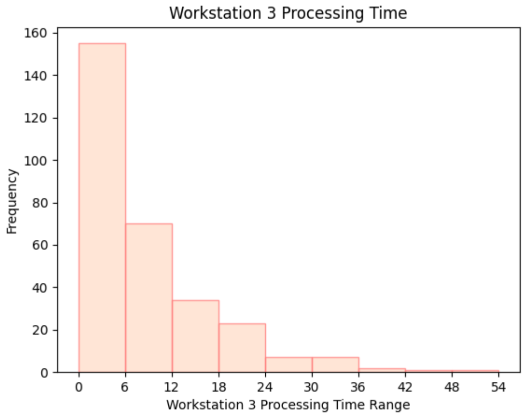


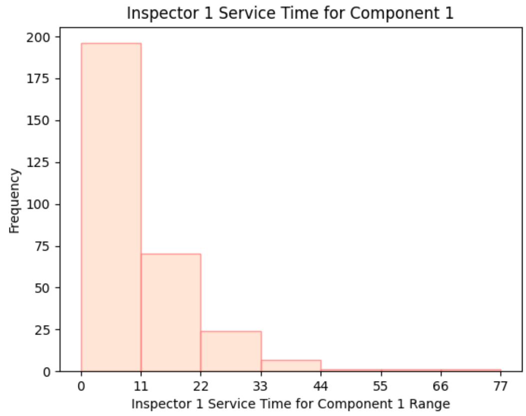
Identify Distribution (Histogram):

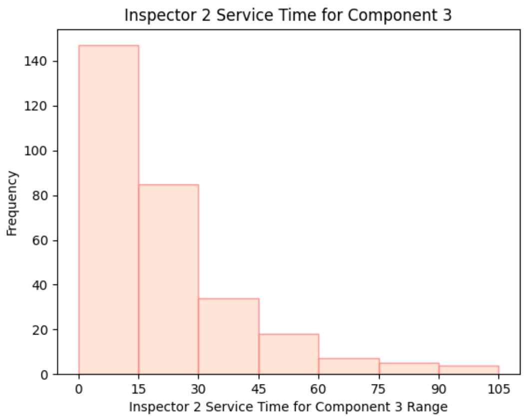
Below will be histograms for the data sets provided.











It can be seen from the distributions that all the datasets follow an exponential distribution.

Evaluate Distributions (Q-Q Plots):

In this section, the Q-Q plots will be provided based on which the evaluation of the data will be done.

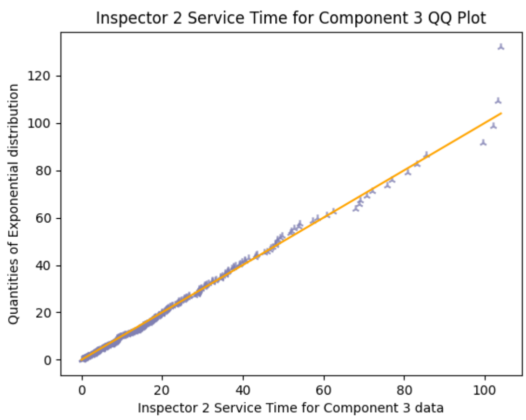
Based on the Q-Q Plot for the exponential distribution of Workstation 1's Processing Time, we can say that "Exponential distribution" is approximately a good distribution as it mostly follows a straight line from the start to the middle. Later for higher values at the ends it diverges.

Based on the Q-Q Plot for the exponential distribution of Workstation 2's Processing Time, we can say that "Exponential distribution" is not a very good distribution overall. This is because it only follows a straight line at the starting quantiles and later diverges from the straight line slightly in the middle and extremely towards the end. But still is a good fit.

Based on the Q-Q Plot for the exponential distribution of Workstation 3's Processing Time, we can say that "Exponential distribution" is a very good distribution overall. It is a better fit than. It follows a straight-line from start to end and barely deviates.

Based on the Q-Q Plot for the Exponential distribution of Inspector 1's Service Time for Component 1, we can say it follows "Exponential distribution" as the points are mostly on a straight linee at the start, middle, and end of the distribution.

Similarly, for Inspectors 2's Service Times for Components 2 and 3, the distribution follows the "Exponential Distribution" pattern as the points are mostly on a straight line at the start and middle of the distribution. For component 3 it is a much better fit as it is only slightly deviating at the end while the rest is on a straight line.



Goodness of Fit (Chi-Square Test):

Workstation 1 Processing Time:

H_0: Random variable is exponentially distribution.

H_1: Random variable is not exponentially distribution.

	range	O	O (new)	E	E (new)	O-E	(O-E)^2/E
0	(0, 4]	183	183	174.155	174.155	8.845047	0.449226
1	(4, 8]	64	64	73.05513	73.05513	-9.05513	1.122376
2	(8, 12]	29	29	30.64542	30.64542	-1.64542	0.088346
3	(12, 16]	12	12	12.85525	12.85525	-0.85525	0.056899
4	(16, 20]	7	12	5.392564	9.00161	2.99838	0.998742
5	(20, 24]	3		2.262092			
6	(24, 28]	1		0.94891			
7	(28, 32]	1		0.398052			
						Sum:	2.715589

Degrees of freedom = k – s – 1 = 5 – 1 – 1 = 3

Level of significance = 0.05

Chi-Square (0.05,3) = 7.81

2.71 < 7.81

H_0 is accepted

Workstation 2 Processing Time:

H_0: Random variable is exponentially distribution.

H_1: Random variable is not exponentially distribution.

	range	O	O (New)	E	E(New)	O-E	(O-E)^2/E
0	(0, 7]	147	147	140.3903	140.3903	6.60971	0.311192
1	(7, 14]	76	76	74.69218	74.69218	1.307822	0.022899
2	(14, 21]	29	29	39.73866	39.73866	-10.7387	2.901929
3	(21, 28]	19	19	21.14225	21.14225	-2.14225	0.217065
4	(28, 35]	9	9	11.24836	11.24836	-2.24836	0.44941
5	(35, 42]	9	9	5.984493	5.984493	3.015507	1.519475
6	(42, 49]	5	11	3.183944	5.779147	5.220853	4.716492
7	(49, 56]	5		1.693961			
8	(56, 63]	1		0.901242			
						Sum:	10.138462

Degrees of freedom = k – s – 1 = 7 – 1 – 1 = 5

Level of significance = 0.05

Chi-Square (0.05,3) = 11.1

10.13 < 11.1

<u>H_0 is accepted</u>

Workstation 3 Processing Time:

H_0: Random variable is exponentially distribution.

H_1: Random variable is not exponentially distribution.

	range	O	O (New)	E	E (New)	O-E	(O-E)^2/E
0	(0, 6]	155	155	148.343	148.343	6.657041	0.298741
1	(6, 12]	70	70	74.99085	74.99085	-4.99085	0.332155
2	(12, 18]	34	34	37.90963	37.90963	-3.90963	0.403202
3	(18, 24]	23	23	19.16421	19.16421	3.835791	0.767748
4	(24, 30]	7	7	9.687958	9.687958	-2.68796	0.745783
5	(30, 36]	7	11	4.89749	9.257559	1.742441	0.327959
6	(36, 42]	2		2.475796			
7	(42, 48]	1		1.251573			
8	(48, 54]	1		0.6327			
						Sum:	2.875588

Degrees of freedom = k – s – 1 = 6 – 1 – 1 = 4

Level of significance = 0.05

Chi-Square (0.05,3) = 9.49

2.87< 9.49

<u>H_0 is accepted</u>

Inspector 1 Service Time for Component 1:

H_0: Random variable is exponentially distribution.

H_1: Random variable is not exponentially distribution.

	range	O	O (New)	E	E (New)	O-E	(O-E)^2/E
0	(0, 11]	196	196	196.2699	196.2699	-0.26992	0.000371
1	(11, 22]	70	70	67.86365	67.86365	2.136351	0.067252
2	(22, 33]	24	24	23.46501	23.46501	0.534993	0.012198
3	(33, 44]	7	10	8.113424	12.22417	2.22417	0. 404684
4	(44, 55]	1		2.805354			
5	(55, 66]	1		0.969999			
6	(66, 77]	1		0.335393			
						Sum:	0.484505

Degrees of freedom = k – s – 1 = 4 – 1 – 1 = 2

Level of significance = 0.05

Chi-Square (0.05,3) = 5.99

0.48 < 5.99

<u>**H_0 is accepted**</u>

Inspector 2 Service Time for Component 2:

H_0: Random variable is exponentially distributed.

H_1: Random variable is not exponentially distributed.

	range	O	O (New)	E	E (New)	O-E	(O-E)^2/E
0	(0, 17]	198	198	199.5547	199.5547	-1.5547	0.012112
1	(17, 34]	74	74	66.81444	66.81444	7.18556	0.772771
2	(34, 51]	20	20	22.37066	22.37066	-2.37066	0.251222
3	(51, 68]	5	8	7.490091	11.1187	3.1187	0.874768
4	(68, 85]	1		2.507815			
5	(85, 102]	1		0.839661			
6	(102, 119]	1		0.281133			
						Sum:	1.910873

Degrees of freedom = k – s – 1 = 4 – 1 – 1 = 2

Level of significance = 0.05

Chi-Square (0.05,3) = 5.99

1.91 < 5.99

<u>H_0 is accepted</u>

Inspector 2 Service Time for Component 2:

H_0: Random variable is exponentially distributed.

H_1: Random variable is not exponentially distributed.

	range	O	O (New)	E	E (New)	O-E	(O-E)^2/E
0	(0, 15]	147	147	154.9928	154.9928	-7.99283	0.412183
1	(15, 30]	85	85	74.91691	74.91691	10.08309	1.357088
2	(30, 45]	34	34	36.21163	36.21163	-2.21163	0.135075
3	(45, 60]	18	18	17.50315	17.50315	0.496849	0.014104
4	(60, 75]	7	7	8.460275	8.460275	-1.46027	0.252049
5	(75, 90]	5	9	4.089335	6.065944	2.934056	1.419183
6	(90, 105]	4		1.976609			
						Sum:	3.589682

Degrees of freedom = k – s – 1 = 6 – 1 – 1 = 4

Level of significance = 0.05

Chi-Square (0.05,3) = 9.49

3.58 < 9.49

<u>H_0 is accepted</u>

Based on the Chi-Square Test conducted for all 6 Datasets we can say that all 6 follow Chi-Square Distribution.

Generate Input Based on Model:

Model Identified: Exponential Distribution

Parameters need: λ

Parameter estimate: $\lambda = 1$

Procedure:

Step 1:

Generate uniform distribution of Sudo Random Number by Linear Congruential Method (Refer 7.3.1 in Textbook).

$X_{i+1} = (a X_i + c) \bmod m$ $\qquad$ $i = 1, 2, 3, 4, \ldots$

Here follow best practices mentioned in the textbook.

m -> a power of 2, as large as possible

- ⇨ $m = 2^{48}$

c -> not equal to zero, relatively prime to m

- ⇨ $c = 2^7 - 1$

a -> 1 + 4k where k is integer

- ⇨ $a = 1 + 4 * 2 = 9$

X_0 any arbitrary value

Step 2:

Convert X_i to the range [0, 1] resulting in a uniform distribution.

$R_i = X_i / m$

Step 3:

Use this random uniform distribution of numbers to get exponential distribution with the following formula (detailed explanation in 8.1.1 Textbook):

$X_i = - \ln(R_i) / \lambda$

(or)

$X_i = - \ln(R_i) * \mu$

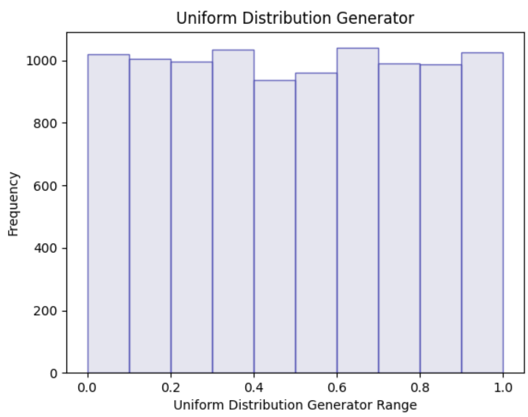
Step 4:

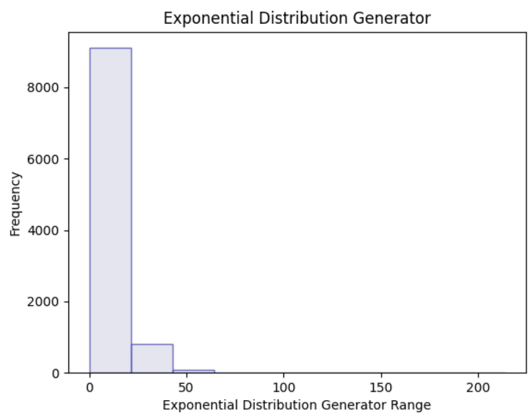
Generate the numbers

Resultant Distribution:

Based on the above-mentioned steps a generator was made.

The output of distribution for Step2, 3 are presented for the generator for the Mean = 8.45





Frequency Test (K-S Test):

H_0: Random variable is Uniform [0,1]

H_1: Random variable is not Uniform [0,1]

Range	O (Observed)	T (Theoretical)	F_O (X)	F_T(X)	| F_O (X) - F_T(X)|
(0.0, 0.1]	1018	1000	1018/10000	1000/10000	18/10000
(0.1, 0.2]	1006	1000	1006/10000	1000/10000	6/10000
(0.2, 0.3]	997	1000	997/10000	1000/10000	3/10000
(0.3, 0.4]	1035	1000	1035/10000	1000/10000	35/10000
(0.4, 0.5]	939	1000	939/10000	1000/10000	61/10000
(0.5, 0.6]	961	1000	961/10000	1000/10000	39/10000
(0.6, 0.7]	1040	1000	1040/10000	1000/10000	40/10000
(0.7, 0.8]	990	1000	990/10000	1000/10000	10/10000
(0.8, 0.9]	987	1000	987/10000	1000/10000	13/10000
(0.9, 1.0]	1027	1000	1027/10000	1000/10000	27/10000

N = 10000

α = 0.05

D = max (| F_O (X) - F_T(X)|)

⇨ D = 61/10000 = 0.0061

D_{10000} 0.05 = 1.36/Sqrt(10000) = 1.36/100 = 0.0136

0.0061 < 0.0136

H_0 is accepted

Test for Autocorrelation:

First 10 random numbers are:

[0.33, 0.04, 0.40, 0.60, 0.44, 0.02, 0.22, 0.01, 0.12, 0.10]

H_0: $\rho_{im} = 0$ if numbers are independent

H_1: ρ_{im} != 0 if numbers are dependent

l = 1

α = 0.05

N = 10

i = 1 (start from the first number)

M = 8

- ⇨ i + (M +1) l <= N
- ⇨ 1 + (M + 1) 1 <= 10
- ⇨ M + 2 <= 10
- ⇨ M <= 8

ρ_{11} = 1/(M +1) [$Sum_{0,\,M}(R_{i+kl}*R_{i+(k+1)l})$] – 0.25

= 1/(8+1) [0.33*0.04 + 0.04*0.40 + 0.40*0.60 + 0.60*0.44 + 0.44*0.02 + 0.02*0.22 + 0.22*0.01 + 0.01*0.12] – 0.25

= 1/9 *[0.5498] – 0.25

= -0.188

$\sigma_{\rho 11}$ = Sqrt(13 * M + 7)/ (12 * (M + 1))

= Sqrt(13 * 8 + 7) / (12 * (8 + 1))

= Sqrt(111) / (108)

= 10.535 /108

= 0.097

Z_0 = - 0.188/0.097 = -1.947

$Z_{0.025}$ = 1.96

-1.947 < 1.96

H_0 is accepted

Code:

```
class Random:

    __seed = None
    __multiplier = 9
    __increment = 2**7 - 1
    __modulus = 2**48
    __last_rand = None

    # Constructor
    def __init__(self, seed=0):
        self.seed = seed
        self.last_rand = seed

    # Returns uniform distribution of the random number
    def random_probability(self):
        self.last_rand = (self.__multiplier * self.last_rand +
self.__increment) % self.__modulus
        return self.last_rand / self.__modulus

    # Returns randomly distributed exponential numbers
    def random_exponential(self, mean=None):
        if (mean is None) or (math.isnan(mean)):
            raise Exception("Mean Not Specified")
        uniform_rand = self.random_probability()
        return -1 * mean * math.log(uniform_rand, math.e)
```

Model Verification and Validation:

Parameters of Interest:

- Facility Throughput/ product throughput = Number of products produced / total simulation time
- Probability each workstation is busy = Number of minutes the workstation was busy/total simulation time
- Average buffer occupancy of each buffer = SUM(Occupancy of buffer each min) / total simulation time
- Probability that each inspector remains “blocked" = Total minutes the inspector was blocked / total simulation time

Peer Review:

Peer review was not done for this as it is against the guidelines for the project submission. But this should be a step in the verification process.

Little’s Law:

For this project, Littles law was tested on Production Run 8.

We found the following through the output of the simulation:

The average arrival rate into the system for Product 1, Product 2, and Product 3 were 0.87, 0.3, and 0.2 respectively.

Summing them up we get a total arrival rate of 0.92

Therefore, the average arrival rate into the system is 92

We also got the average waiting time for Workstations 1,2,3 as 42%, 2% 3% respectively, thereby getting an average wait time of 45%

Therefore, the average waiting time = 45%

The average number of products produced by workstations 1,2,3 is 38, 2, 2 respectively.

Total to 42.

Therefore 42 ~ 45 * 0.92

= 42 ~ 41.4

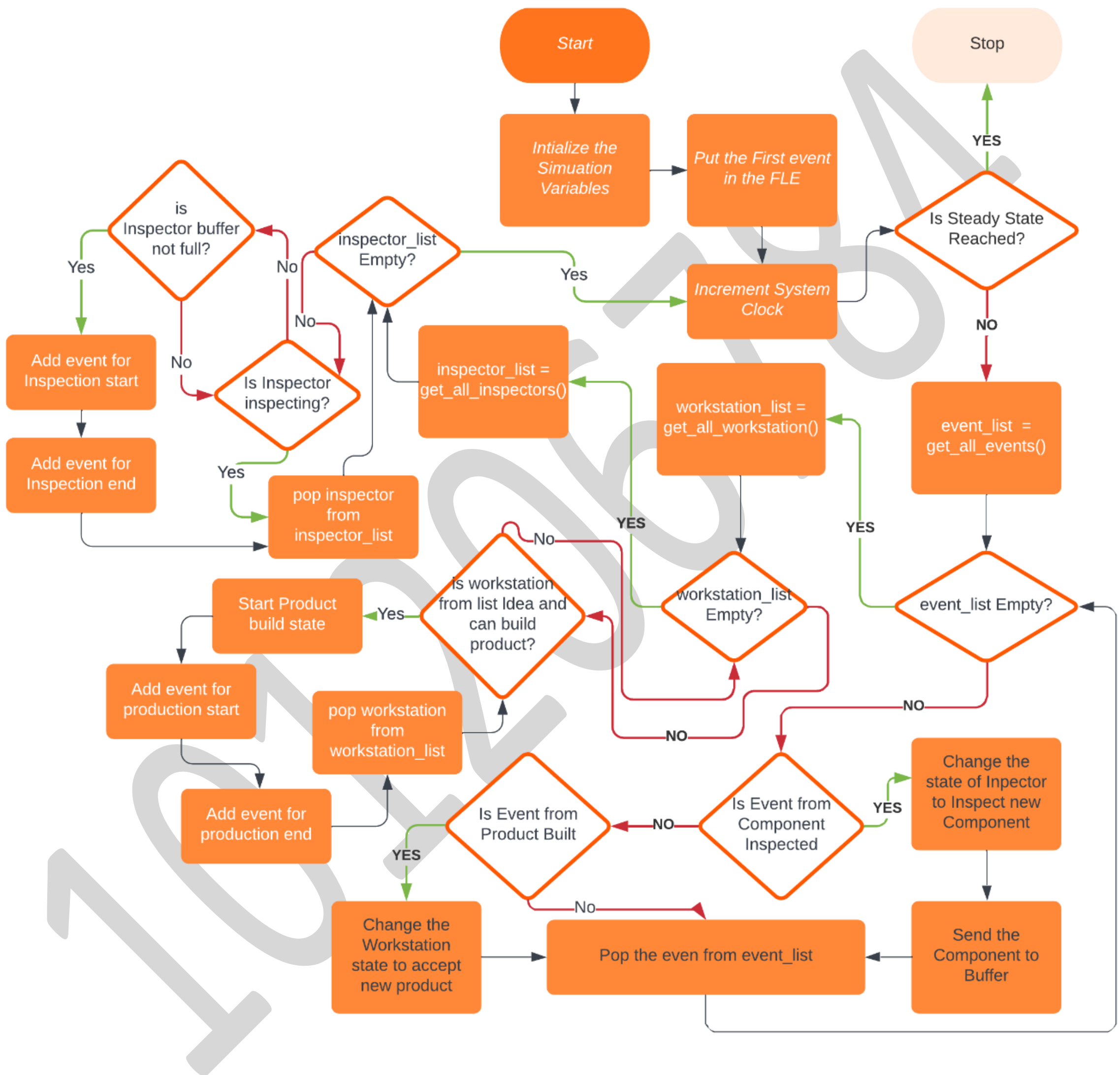
Therefore, the little laws hold.

Also, the number of components 1 entering the system = 48 which matches the number of output products that could be produced.

Similarly for Component 2, 4 components entered the system, and 2 products were produced leaving 2 in the buffer. Same for Component 3.

Operational Model:

The flow chart for the model is given to verify it.



Class and object diagrams are also provided below.

Other verifications like the histogram and QQ are provided in the previous sections. Please refer there.

Trial Run:

The simulation was run for 899.999 minutes which produced the flowing result:

Product 1 throughput: 0.08777777777777777

Product 2 throughput: 0.0033333333333333335

Product 3 throughput: 0.0022222222222222222

Workstation 1 busy probability: 0.42282888888888887

Workstation 2 busy probability: 0.022838888888888888

Workstation 3 busy probability: 0.037102222222222224

Workstation 1 wait time: 4817.037974683544

Workstation 2 wait time: 6851.666666666667

Workstation 3 wait time: 16696.0

Workstation 1 Component count: 79

Workstation 2 Component count: 3

Workstation 3 Component count: 2

Buffer 1 average occupancy: 0.13815222222222223

Buffer 2 average occupancy: 0.22326666666666667

Buffer 3 average occupancy: 0.44434666666666667

Buffer 4 average occupancy: 0.00544

Buffer 5 average occupancy: 1.5369333333333333

Inspector 1 block probability: 0.0

Inspector 2 block probability: 0.8402122222222222

Validation:

As mentioned by the professor this part is hard to do.

But the alternative is to check if the model is working as it should by checking for the initialization phase.

And checking for face validity. This the model does satisfy.

The Histograms and QQ plot also validate the simulation.

Production Runs and Analysis:

Production Run 1:

Stop point = 999.999 min.

result:

Product 1 throughput: 0.088

Product 2 throughput: 0.003

Product 3 throughput: 0.002

Workstation 1 busy probability: 0.413563

Workstation 2 busy probability: 0.020555

Workstation 3 busy probability: 0.033392

Buffer 1 average occupancy: 0.124337

Buffer 2 average occupancy: 0.350719

Buffer 3 average occupancy: 0.336121

Buffer 4 average occupancy: 0.104896

Buffer 5 average occupancy: 1.436362

Inspector 1 block probability: 0.0

Inspector 2 block probability: 0.856191

Production Run 2:

Stop point = 499.999 min.

result:

Product 1 throughput: 0.08

Product 2 throughput: 0.0

Product 3 throughput: 0.0

Workstation 1 busy probability: 0.24756

Workstation 2 busy probability: 0.0

Workstation 3 busy probability: 0.0

Buffer 1 average occupancy: 0.05556

Buffer 2 average occupancy: 0.0

Buffer 3 average occupancy: 0.6952

Buffer 4 average occupancy: 0.0

Buffer 5 average occupancy: 0.0

Inspector 1 block probability: 0.0

Inspector 2 block probability: 0.0

Production Run 3:

Stop point = 99.999 min.

result:

Product 1 throughput: 0.1

Product 2 throughput: 0.0

Product 3 throughput: 0.0

Workstation 1 busy probability: 0.24609

Workstation 2 busy probability: 0.0

Workstation 3 busy probability: 0.0

Buffer 1 average occupancy: 0.09514

Buffer 2 average occupancy: 0.28611

Buffer 3 average occupancy: 0.0

Buffer 4 average occupancy: 0.0

Buffer 5 average occupancy: 0.0

Inspector 1 block probability: 0.0

Inspector 2 block probability: 0.0

Production Run 4:

Stop point = 799.999 min.

result:

Product 1 throughput: 0.09125

Product 2 throughput: 0.0037500000000000003

Product 3 throughput: 0.0

Workstation 1 busy probability: 0.4063125

Workstation 2 busy probability: 0.02569375

Workstation 3 busy probability: 0.02747875

Buffer 1 average occupancy: 0.13366625

Buffer 2 average occupancy: 0.1883975

Buffer 3 average occupancy: 0.42015375

Buffer 4 average occupancy: 0.0

Buffer 5 average occupancy: 1.71247625

Inspector 1 block probability: 0.0

Inspector 2 block probability: 0.82023875

Production Run 5:

Stop point = 699.999 min.

result:

Product 1 throughput: 0.08571428571428572

Product 2 throughput: 0.004285714285714285

Product 3 throughput: 0.0

Workstation 1 busy probability: 0.3869457142857143

Workstation 2 busy probability: 0.029364285714285714

Workstation 3 busy probability: 0.0

Buffer 1 average occupancy: 0.10047

Buffer 2 average occupancy: 0.07245428571428572

Buffer 3 average occupancy: 0.48017857142857145

Buffer 4 average occupancy: 0.0

Buffer 5 average occupancy: 1.702807142857143

Inspector 1 block probability: 0.0

Inspector 2 block probability: 0.7945585714285714

Production Run 6:

Stop point = 899.999 min.

result:

Product 1 throughput: 0.08777777777777777

Product 2 throughput: 0.0033333333333333333

Product 3 throughput: 0.0022222222222222222

Workstation 1 busy probability: 0.42282888888888887

Workstation 2 busy probability: 0.022838888888888888

Workstation 3 busy probability: 0.037102222222222224

Buffer 1 average occupancy: 0.13815222222222223

Buffer 2 average occupancy: 0.22326666666666667

Buffer 3 average occupancy: 0.44434666666666667

Buffer 4 average occupancy: 0.00544

Buffer 5 average occupancy: 1.5369333333333333

Inspector 1 block probability: 0.0

Inspector 2 block probability: 0.8402122222222222

Notes:

The number of replications is enough as we can get the initialization phase from these replications. One more thing is that these replications produce consistent results for the output.

Initialization Phase:

The initialization Phase for the simulation is between 800 to 900 minutes for product 3 as that is the minimum amount of time it takes for the system to produce any output for product 3. For product 2 the initialization phase is between 500 and 600 minutes as it minimum amount of time it took to build at least one quantity of that product. For Product 1 the initialization phase is between 50 min and 100 min.

Confidence Interval:

The confidence interval is given by the formula:

$$CI = \bar{x} \pm z \cdot \frac{s}{\sqrt{n}}$$

Where x is the mean, z-value for the confidence level, n is the sample size, and s is the standard deviation.

Inspector 1:

X = 10.35

Z = 1.96 for 95% confidence

S = 9.78

n = 300

Therefore CI = [9.24,11.45]

Inspector 2 for Component 2:

X = 15.53

Z = 1.96 for 95% confidence

S = 14.68

n = 300

Therefore CI = [13.87,17.19]

Inspector 2 for Component 3:

X = 20.63

Z = 1.96 for 95% confidence

S = 19.85

n = 300

Therefore CI = [18.38,22.87]

Workstation 1:

X = 4.60

Z = 1.96 for 95% confidence

S = 4.75

n = 300

Therefore CI = [4.60,5.13]

Workstation 2:

X = 11.09

Z = 1.96 for 95% confidence

S = 11.84

n = 300

Therefore CI = [9.66,12.42]

Workstation 3:

X = 8.79

Z = 1.96 for 95% confidence

S = 8.65

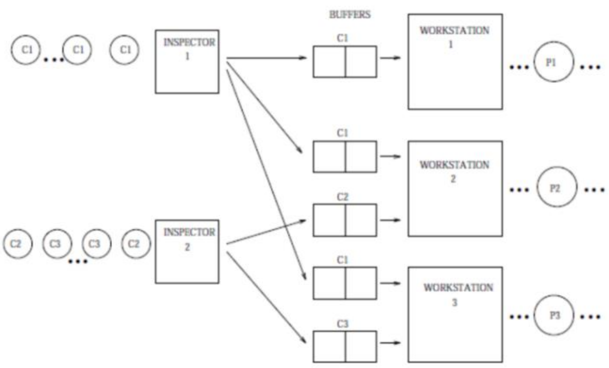
n = 300

Therefore CI = [7.81, 9.76]

Note: I am guessing he wants us to do more, but that part is missing the project document so can't help it.

Alternative Operation Policy:

Alternative Conceptual Model:



A manufacturing facility assembles three different types of products (P1, P2, P3) and has different components (C1, C2, C3) as follows:

- P1: C1
- P2: C1, C2
- P3: C1, C3

Two inspectors (I1, I2) clean and repair the components as follows:

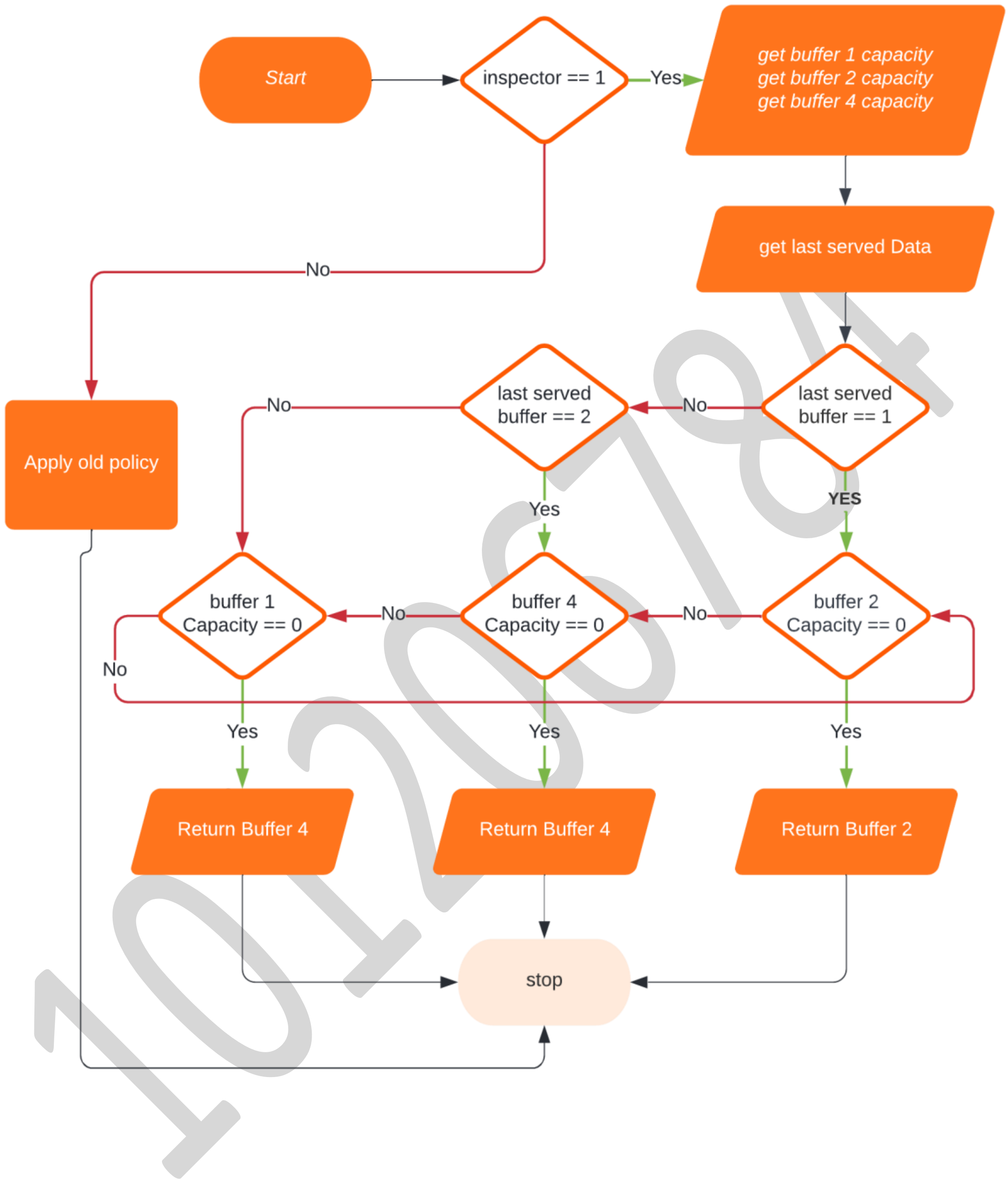
- I1: C1
- I2: C2, C3 (Randomly)

The inspectors will never have to wait for components. There is an infinite inventory of them always immediately available.

There are three workstations in the facility, named W1, W2, and W3, which assemble products P1, P2, and P3, respectively. After the components pass inspection, they are sent to their respective workstations. Each workstation has a buffer capacity of two components, with one buffer available for each of the component types needed. A product can begin being assembled only when components of all types required are available. If all workstation buffers for a specific type of component are full, the corresponding inspector who finished inspecting a component with the same type is considered “blocked" until there is an opening, at which time the inspector can resume processing and sending components of that type.

In the present mode of operation, Inspector 1 routes components C1 to the buffer with the smallest number of components in waiting (i.e., a routing policy according to the shortest queue). **In the case of a tie, apply Round Robin to make sure only Workstation 1 is not prioritized**.

Alternative Design Flow Chart:



Alternative Design Policy Production Run:

Stop point = 899.999 min.

result:

Product 1 throughput: 0.05

Product 2 throughput: 0.003

Product 3 throughput: 0.014

Workstation 1 busy probability: 0.224

Workstation 2 busy probability: 0.022

Workstation 3 busy probability: 0.136

Buffer 1 average occupancy: 0.059

Buffer 2 average occupancy: 0.851

Buffer 3 average occupancy: 0.055

Buffer 4 average occupancy: 0.724

Buffer 5 average occupancy: 0.173

Inspector 1 block probability: 0.231

Inspector 2 block probability: 0.591

Comparison:

	Original Policy	Alternative Policy
Product 1 throughput	0.087	0.050
Product 2 throughput	0.003	0.003
Product 3 throughput	0.002	0.014
Workstation 1 busy probability	0.422	0.224
Workstation 2 busy probability	0.022	0.022
Workstation 3 busy probability	0.037	0.136
Buffer 1 average occupancy	0.138	0.059
Buffer 2 average occupancy	0.223	0.851
Buffer 3 average occupancy	0.444	0.055
Buffer 4 average occupancy	0.005	0.724
Buffer 5 average occupancy	1.536	0.173
Inspector 1 block probability	0.000	0.231
Inspector 2 block probability	0.840	0. 591

Consequences of Alternative Policy:

With the change in this policy, we notice a tremendous increase in the throughput of product 3 by 700%. Although there was a decrease in the throughput in product 1 by 30% which was anticipated. Overall, this new policy increased the throughput.

There is also an increase in the amount of time the workstations remain busy i.e., producing products. This is a move in the right direction.

There are more components in buffers 2 and 4 which is the ideal case. This does reduce the number of components in buffer 1 but this is good as we produce more products.

Finally, the block time for the Inspectors is reduced which is good.

Conclusion:

In conclusion, I would say that the original policy, in theory, is ideal but in implementation only sends components to workstation 1. The alternative to this is round robin which is a way better solution for inspector 1 only as it produces more products 2 and 3.

Code:

```
import pandas as pd
from random import random
import os
import enum
from Component import Component
from Inspector import Inspector
from Buffer import Buffer
from Workstation import Workstation


# This class acts as the future event list
class FEL:
    __queue = None  # queue

    # constructor
    def __init__(self):
        self.__queue = {}

    # adds events to FEL
    def put_event(self, event, time):
        if time not in self.__queue:
            self.__queue[time] = []
        self.__queue[time].append(event)

    # retrieve event list by time
    def get_events(self, time) -> list:
        if time not in self.__queue:
            return []
        else:
            return self.__queue[time]

    # prints FEL
    def print(self):
        print(self.__queue)


# Enumerated class to hold event types
class EventType(enum.Enum):
    arrival = 0
    departure = 1
    delay = 2


# Enumerated class to hold on which item the event occurred
```

```
class ItemType(enum.Enum):
    component = 0
    product = 1


# Class that holds the simulation environment and runs the main simulation
code
class Simulator:
    __inspectors = []  # list of inspectors
    __buffers = []  # list of buffers
    __workstations = []   # list of workstations
    sys_clock = 0   # system clock
    __last_buffer = 4  # buffer tracking for policy 2

    # dataset file locations
    base_path = os.path.realpath(__file__) + os.sep + os.pardir + os.sep +
os.pardir + os.sep + "dataset" + os.sep
    WS1_TIME_DATA_FILE = base_path + "ws1.dat"
    WS2_TIME_DATA_FILE = base_path + "ws2.dat"
    WS3_TIME_DATA_FILE = base_path + "ws3.dat"
    I11_TIME_DATA_FIlE = base_path + "servinsp1.dat"
    I22_TIME_DATA_FIlE = base_path + "servinsp22.dat"
    I23_TIME_DATA_FIlE = base_path + "servinsp23.dat"

    # Future Event List Object
    FEL = FEL()

    # Constructor
    def __init__(self):
        self.sys_clock = 0

        # init dataset queues
        self.I11_queue = self.__get_list_from_file(self.I11_TIME_DATA_FIlE)
        self.I22_queue = self.__get_list_from_file(self.I22_TIME_DATA_FIlE)
        self.I23_queue = self.__get_list_from_file(self.I23_TIME_DATA_FIlE)
        self.WS1_queue = self.__get_list_from_file(self.WS1_TIME_DATA_FILE)
        self.WS2_queue = self.__get_list_from_file(self.WS2_TIME_DATA_FILE)
        self.WS3_queue = self.__get_list_from_file(self.WS3_TIME_DATA_FILE)

        # init inspectors
        i1 = Inspector(i_type=1)
        i2 = Inspector(i_type=2)
        self.__inspectors.append(i1)
        self.__inspectors.append(i2)

        # init buffers
        b1 = Buffer(priority=3)
        b2 = Buffer(priority=2)
        b3 = Buffer(priority=2)
        b4 = Buffer(priority=1)
        b5 = Buffer(priority=1)
        self.__buffers.append(b1)
        self.__buffers.append(b2)
        self.__buffers.append(b3)
        self.__buffers.append(b4)
        self.__buffers.append(b5)

        # init workstations
```

```
        w1_buffers = [b1]
        w2_buffers = [b2, b3]
        w3_buffers = [b4, b5]
        w1 = Workstation(1, w1_buffers)
        w2 = Workstation(2, w2_buffers)
        w3 = Workstation(3, w3_buffers)
        self.__workstations.append(w1)
        self.__workstations.append(w2)
        self.__workstations.append(w3)

    # returns a queue of data from from data file
    @staticmethod
    def __get_list_from_file(file_name) -> list:
        df = pd.read_csv(file_name, header=None)
        time_queue = df.to_numpy().flatten().tolist()
        return time_queue

    # returns the next processing time for workstations
    def __get_workstation_next_processing_time(self, w_id) -> int:
        processing_time = None
        if w_id == 1:
            if len(self.WS1_queue) == 0:
                return None
            processing_time = self.WS1_queue.pop(0)
        if w_id == 2:
            if len(self.WS2_queue) == 0:
                return None
            processing_time = self.WS2_queue.pop(0)
        if w_id == 3:
            if len(self.WS3_queue) == 0:
                return None
            processing_time = self.WS3_queue.pop(0)
        return self.__convert_time_to_int(processing_time)

    # returns a component C1 for Inspector 1
    def __get_i1_next_component(self) -> tuple:
        if len(self.I11_queue) == 0:
            return None, None
        inspection_time = self.I11_queue.pop(0)
        component = Component(1)
        return self.__convert_time_to_int(inspection_time), component

    # returns randomly a component to Inspector 2
    def __get_i2_next_component(self) -> tuple:
        if len(self.I22_queue) == 0 and len(self.I23_queue) == 0:
            return None, None
        elif len(self.I22_queue) == 0:
            inspection_time = self.I23_queue.pop(0)
            component = Component(3)
        elif len(self.I23_queue) == 0:
            inspection_time = self.I22_queue.pop(0)
            component = Component(2)
        elif random() <= 0.5:
            inspection_time = self.I22_queue.pop(0)
            component = Component(2)
        else:
            inspection_time = self.I23_queue.pop(0)
            component = Component(3)
```

```
        return self.__convert_time_to_int(inspection_time), component

    # increment system clock
    def __inc_clock(self):
        self.sys_clock += 1

    # converts float dataset to integer dataset
    @staticmethod
    def __convert_time_to_int(time) -> int:
        return int(1000 * time)

    # returns the buffer_id based on component for policy 1
    def __send_to_buffer_policy_1(self, inspector_id, component) -> int:
        if inspector_id == 1:
            b1_capacity = self.__buffers[0].get_capacity()
            b2_capacity = self.__buffers[1].get_capacity()
            b4_capacity = self.__buffers[3].get_capacity()

            if b1_capacity <= b2_capacity and b1_capacity <= b4_capacity:
                return 1
            if b2_capacity <= b4_capacity:
                return 2
            return 4
        elif inspector_id == 2:
            c_type = component.get_type()
            if c_type == 2:
                return 3
            if c_type == 3:
                return 5

    # returns the buffer_id based on component for policy 2
    def __send_to_buffer_policy_2(self, inspector_id, component) -> int:
        b1_capacity = self.__buffers[0].get_capacity()
        b2_capacity = self.__buffers[1].get_capacity()
        b4_capacity = self.__buffers[3].get_capacity()

        # apply round robin if any buffer is empty for inspector 1
        if (inspector_id == 1) and ((b1_capacity == 0) or (b2_capacity ==
0) or (b4_capacity == 0)):
            if self.__last_buffer == 1:
                self.__last_buffer = 2
                if b2_capacity == 0:
                    self.__last_buffer = 2
                    return 2
                elif b4_capacity == 0:
                    return 4
                else:
                    return 1

            elif self.__last_buffer == 2:
                self.__last_buffer = 4
                if b4_capacity == 0:
                    return 4
                elif b1_capacity == 0:
                    return 1
                else:
                    return 2
            elif self.__last_buffer == 4:
```

```
                self.__last_buffer = 1
                if b1_capacity == 0:
                    return 1
                elif b2_capacity == 0:
                    return 2
                else:
                    return 4
            else:
                raise Exception("Unknown Last buffer")
        # else do the old policy 1
        else:
            buffer = self.__send_to_buffer_policy_1(inspector_id,
component)
            return buffer

    # returns the buffer_id based on component
    def __send_to_buffer(self, inspector_id, component) -> int:
        buffer = self.__send_to_buffer_policy_2(inspector_id, component)
        return buffer

    # main function running the simulation
    def run_simulation(self, sim_time=499999):
        end_sim_flag = False

        products_count = [0, 0, 0]
        workstations_busy_count = [0, 0, 0]
        buffer_occupancy_count = [0, 0, 0, 0, 0]
        inspector_block_count = [0, 0]

        # get first components to Inspector 1 & 2
        (new_I1_time, I1_component) = self.__get_i1_next_component()
        (new_I2_time, I2_component) = self.__get_i2_next_component()

        # add inspection events to FEL
        self.FEL.put_event((EventType.arrival, ItemType.component,
I1_component), 0)
        self.FEL.put_event((EventType.arrival, ItemType.component,
I2_component), 0)
        # perform inspection
        self.__inspectors[0].inspect_component(I1_component, new_I1_time)
        self.__inspectors[1].inspect_component(I2_component, new_I2_time)
        # add completion events to FLE
        self.FEL.put_event((EventType.departure, ItemType.component,
I1_component), new_I1_time)
        self.FEL.put_event((EventType.departure, ItemType.component,
I2_component), new_I2_time)

        self.__inc_clock()  # finish first cycle

        # run simulation till specified time
        while self.sys_clock <= sim_time:
            # get all current events
            events = self.FEL.get_events(self.sys_clock)

            # loop over all events
            for event in events:

                # extract event info
```

```
                e_type = event[0]
                i_type = event[1]

                # check if the event is -- Component finished being
inspected
                if (e_type == EventType.departure) and (i_type ==
ItemType.component):
                    component = event[2]
                    # get the inspector for the event
                    inspector_id = 1 if component.get_type() == 1 else 2
                    # get the buffer for the event
                    buffer_id =
self.__send_to_buffer(inspector_id=inspector_id, component=component)

                    #   perform the actions:
                    #       inspector sends component
                    #       insert component to buffer
                    self.__inspectors[inspector_id - 1].send_component()
                    self.__buffers[buffer_id -
1].insert_component(component=component)
                    print(f"Log:\tFinished Comp: {component.get_type()}")

                # check if the event is -- Product built
                if (e_type == EventType.departure) and (i_type ==
ItemType.product):
                    product = event[2]
                    # get the workstation for the event
                    workstation_id = product.get_type()
                    product_type = product.get_type()

                    #   perform the actions:
                    #       finish building product
                    self.__workstations[workstation_id -
1].finish_building_product()

                    # increment product production count
                    products_count[product_type - 1] += 1

                    print(f"Log:\tProduced Product: {product_type}")

            # Check all workstations
            for workstation in self.__workstations:
                can_dequeue = workstation.can_dequeue_buffers()

                workstation_id = workstation.get_type()

                # increment workstation busy time if it is busy
                if workstation.is_making_product:
                    workstations_busy_count[workstation_id - 1] += 1

                # if workstation is ideal and can build products build
product
                if (not workstation.is_making_product) and can_dequeue:
                    processing_time =
self.__get_workstation_next_processing_time(w_id=workstation_id)

                    # start building product
                    is_building, product =
```

```
workstation.start_building_product(processing_time)

                    if is_building:
                        # add the product building events
                        self.FEL.put_event((EventType.arrival,
ItemType.product, product), self.sys_clock)
                        self.FEL.put_event((EventType.departure,
ItemType.product, product), self.sys_clock +
                                           processing_time)
                        print(f"Log:\tStarted building Product:
{workstation_id}")

            # Check all inspectors
            for inspector in self.__inspectors:
                inspector_id = inspector.get_type()

                # Check if inspector is not inspecting
                if not self.__inspectors[inspector_id - 1].is_inspecting:
                    new_component = None
                    new_time = None
                    if inspector_id == 1:
                        new_time, new_component =
self.__get_i1_next_component()
                    elif inspector_id == 2:
                        new_time, new_component =
self.__get_i2_next_component()

                    if new_component is None:
                        print("Component List Finished")
                        end_sim_flag = True
                        break

                    buffer_id =
self.__send_to_buffer(inspector_id=inspector_id, component=new_component)

                    # if inspector buffer are not full inspect
                    if not self.__buffers[buffer_id - 1].is_full:
                        # create events for new component
                        self.FEL.put_event((EventType.arrival,
ItemType.component, new_component), self.sys_clock)
                        self.FEL.put_event((EventType.departure,
ItemType.component, new_component),
                                           self.sys_clock + new_time)

                        # perform actions on the new component
                        self.__inspectors[inspector_id -
1].inspect_component(new_component, new_I1_time)
                        print("Log: \tStarted Inspecting Component:",
new_component.get_type())

                    # inspector buffers are full put the component back on
hold
                    # and increment inspector block time
                    else:
                        new_component_type = new_component.get_type()
                        if new_component_type == 1:
                            self.I11_queue.insert(0, new_time)
                        elif new_component_type == 2:
```

```
                        self.I22_queue.insert(0, new_time)
                    elif new_component_type == 3:
                        self.I23_queue.insert(0, new_time)

                    # increment inspector block time
                    inspector_block_count[inspector_id - 1] += 1

            # Increment buffer capacity count
            for index, buffer in enumerate(self.__buffers):
                buffer_occupancy_count[index] += buffer.get_capacity()

            if end_sim_flag:
                break

            # increment system Clock
            self.__inc_clock()

        #   ----------- Simulation Ends -----------------

        # Print FEL
        # self.FEL.print()

        products_throughput = []
        for product_count in products_count:
            try:

products_throughput.append(product_count/(self.sys_clock/1000))
            except ZeroDivisionError:
                products_throughput.append(None)

        probability_workstation_busy = []
        avg_wait_time = []
        for index, busy_count in enumerate(workstations_busy_count):
            probability_workstation_busy.append(busy_count/self.sys_clock)
            try:
                avg_wait_time.append(busy_count/products_count[index]/1000)
            except ZeroDivisionError:
                avg_wait_time.append(None)

        avg_buffer_occupancy = []
        for occupancy_count in buffer_occupancy_count:
            avg_buffer_occupancy.append(occupancy_count/self.sys_clock)

        probability_inspector_blocked = []
        for block_time in inspector_block_count:
            probability_inspector_blocked.append(block_time/self.sys_clock)

        for index, throughput in enumerate(products_throughput):
            print(f"Product {index + 1} throughput: {throughput}")
        for index, probability_busy in
enumerate(probability_workstation_busy):
            print(f"Workstation {index + 1} busy probability:
{probability_busy}")
        for index, wait_time in enumerate(avg_wait_time):
            print(f"Workstation {index + 1} wait time: {wait_time}")
        for index, product_count_value in enumerate(products_count):
            print(f"Workstation {index + 1} products produced:
{product_count_value}")
```

```
        for index, occupancy in enumerate(avg_buffer_occupancy):
            print(f"Buffer {index + 1} average occupancy: {occupancy}")
        for index, probability_blocked in
enumerate(probability_inspector_blocked):
            print(f"Inspector {index + 1} block probability:
{probability_blocked}")
```