**CMPE412-S01-2023.2 Computer Simulation**

**PROJECT 2**

**MANUFACTURING**

**By:**

Nouhayla HARCHI

20201701082

1. **Introduction**

This project involves developing a discrete-event simulation to model a high-volume automotive parts manufacturing line. Initially, the focus is on a single product line to simplify development. Then there is a bonus challenge includes extending the simulation to handle multiple product types, each with unique manufacturing requirements.

1. **Document the design**

Based on the instruction we have been given, this project’s objectives are:   
To develop basic functionalities for discrete-event simulation, including event scheduling, time progression, and state updates. The system modelling requires defining raw materials, intermediate parts, and finished products while detailing operations at each production stage, such as loading, machining, assembling, inspecting, and packaging, and identifying and modelling resources like machines and operators.

The data requirements include establishing times for each machine and operation based on hypothetical but realistic estimates, configuring the frequency of raw material inputs, determining typical failure rates and maintenance times for machines, and programming shift patterns and worker allocations.

For simulation implementation involves developing a system to manage events like the start and end of processes and resource breakdowns, implementing a simulation clock to advance to the next event, and allowing users to modify input variables like machine count and shift timing.

For experimentation and analysis, run basic scenarios focusing on a single product type to establish baseline performance, adjust key variables to observe effects on throughput and identify bottlenecks, collect data, analyse performance, and generate recommendations.

1. **Implementation details**

**CODE SNIPPET:**def loading(env, task\_name):  
 log\_event('start', env.now, task\_name)  
 print(f"{task\_name} started at {env.now} minutes")  
 yield env.timeout(LOADING\_TIME)  
 log\_event("finish", env.now, task\_name)  
 print(f"{task\_name} finished at {env.now} minutes")  
  
  
  
def machining(env, raw\_material\_queue, machining\_queue):  
 while True:  
 #starting time  
 yield raw\_material\_queue.get()  
 log\_event('start', env.now, 'Machining')  
 print(f"Machining will start at {env.now} minutes")  
 #finishing time  
 yield env.timeout(MACHINING\_TIME)  
 machining\_queue.put(env.now)  
 log\_event('finish', env.now, 'Machining')  
 print(f"Machining finished at {env.now} minutes")  
  
  
def assembly(env, machining\_queue, assembly\_queue):  
 while True:  
 yield machining\_queue.get()  
 log\_event('start', env.now, 'Assembly')  
 print(f"Assembly started at {env.now} minutes")  
 yield env.timeout(ASSEMBLY\_TIME)  
 assembly\_queue.put(env.now)  
 log\_event('finish', env.now, 'Assembly')  
 print(f"Assembly finished at {env.now} minutes")  
  
  
def quality\_control(env, assembly\_queue, qc\_queue):  
 while True:  
 yield assembly\_queue.get()  
 log\_event('start', env.now, 'Quality Control')  
 print(f"Quality control start is at {env.now} minutes")  
 yield env.timeout(QUALITY\_CONTROL\_TIME)  
 qc\_queue.put(env.now)  
 log\_event('finish', env.now, 'Quality Control')  
 print(f"Quality control finished at {env.now} minutes")  
  
  
def package(env, qc\_queue, packaging\_queue):  
 while True:  
 yield qc\_queue.get()  
 log\_event('start', env.now, 'Packaging')  
 print(f"Packaging start is at {env.now} minutes")  
 yield env.timeout(PACKAGING\_TIME)  
 packaging\_queue.put(env.now)  
 log\_event('finish', env.now, 'Packaging')  
 print(f"Packaging finished at {env.now} minutes")  
 print(f"Packaging finished at {env.now} minutes")  
  
  
#the main simulation function  
def run\_simulation():  
 env = simpy.Environment()  
  
 raw\_material\_queue = simpy.Store(env)  
 machining\_queue = simpy.Store(env)  
 assembly\_queue = simpy.Store(env)  
 qc\_queue = simpy.Store(env)  
 packaging\_queue = simpy.Store(env)  
  
 env.process(raw\_material\_generator(env, raw\_material\_queue))  
 env.process(machining(env, raw\_material\_queue, machining\_queue))  
 env.process(assembly(env, machining\_queue, assembly\_queue))  
 env.process(quality\_control(env, assembly\_queue, qc\_queue))  
 env.process(package(env, qc\_queue, packaging\_queue))  
  
 env.run(until=SIMU\_TIME)  
  
 #then visualizing every thing using a gantt chart  
 visualize\_events()

There are further explanations in the code.

1. **Findings**

Since python is a strong language regarding visualizing data and creating organizing charts. I thought it would better to visualize the timing and schedule of each operation.

A graph with green rectangles

Description automatically generated

*FIGURE- GANTT CHART VISUALIZING THE STEPS*

As we can see in the figure above, after 5 simulations, we got this result. We have the machining that starts first then throughout it the assembly starts at the same controlling the quality and then the packaging come last. The inspecting and loading operations aren’t here but they are still present in the back of each simulation.

1. **Recommendations & Conclusion**

To conlcude, developing such a simulation using a programming language line Python, Java is important for optimizing manufacturing processes by modelling raw materials, operations, and resources. The simulation should mainly implement event management, time progression, and state updates, by allowing for easier modification of input variables such as machine count and shift timing. And considering some recommendation like starting with small numbers of simulation time or with a single product type to establish a base before adding complexity makes sure of a better output.