**19CSE312-DISTRIBUTED SYSTEMS**

CASE STUDY



**Topic**: Chandy Misra Hass AND Model Algorithm

**TEAM DETAILS**

MANTHINI MEHER VARDHAN - CB.EN.U4CSE21034

MURABOYINA SRIKAR - CB.EN.U4CSE21062

NERELLA GEETHA KRISHNA - CB.EN.U4CSE21040

TANGUDU HARSHA VARDHAN - CB.EN.U4CSE21062

1. **CODE:**

import networkx as nx

import matplotlib.pyplot as plt

class Proc:

def \_init\_(self):

self.depProc = []

self.totDepProcs = 0

self.wait = False

def deadlock\_detect(processes, probeProcess):

global gDeadlock

visited = [False] \* len(processes)

stack = [probeProcess]

while stack:

current\_proc = stack.pop()

if not visited[current\_proc]:

visited[current\_proc] = True

for neighbor in processes[current\_proc].depProc:

if not visited[neighbor]:

stack.append(neighbor)

else:

gDeadlock = True

def visualize\_wait\_graph(processes, wait\_graph):

G = nx.DiGraph()

labels = {}

colors = []

for i, proc in enumerate(processes):

G.add\_node(f'P{i}')

labels[f'P{i}'] = f'P{i}'

if proc.wait:

colors.append('red') # If the process is waiting, color it red

else:

colors.append('green') # Otherwise, color it green

for i in range(len(wait\_graph)):

for j in range(len(wait\_graph[i])):

if wait\_graph[i][j] == 1:

G.add\_edge(f'P{i}', f'P{j}')

pos = nx.spring\_layout(G)

nx.draw(G, pos, node\_color=colors, with\_labels=True)

nx.draw\_networkx\_labels(G, pos, labels=labels)

plt.show()

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

gDeadlock = False

nProcs = int(input("Enter the number of processes: "))

if nProcs > 1:

waitGraph = [[0] \* nProcs for \_ in range(nProcs)]

process = [Proc() for \_ in range(nProcs)]

print("Input the wait graph:\n")

for i in range(nProcs):

print(f"Process P{i} dependencies:")

for j in range(nProcs):

if i == j:

continue

validInput = False

while not validInput:

tmp = input(f"Is process P{i} waiting for P{j}? (Y/N): ")

if tmp.lower() == 'y':

validInput = True

waitGraph[i][j] = 1

process[i].depProc.append(j)

elif tmp.lower() == 'n':

validInput = True

waitGraph[i][j] = 0

else:

print("Invalid input. Please select either (Y/N)")

process[i].totDepProcs = len(process[i].depProc)

while True:

probeProcess = int(input(f"\nInput deadlock detection trigger process (0 - {nProcs - 1}): "))

if 0 <= probeProcess < nProcs:

break

else:

print(f"Please enter a value between 0 and {nProcs - 1}")

print("\n====================================================\n")

print("Wait-for Graph Process Dependencies:\n")

for i, proc in enumerate(process):

print(f"P{i} -> {' '.join(f'P{dep}' for dep in proc.depProc)}")

print("\nStarting Deadlock Detection...\n")

deadlock\_detect(process, probeProcess)

print("\nVisualizing Wait-for Graph...\n")

visualize\_wait\_graph(process, waitGraph)

if gDeadlock:

print("\*")

print("\* DEADLOCK DETECTED \*")

print("\*")

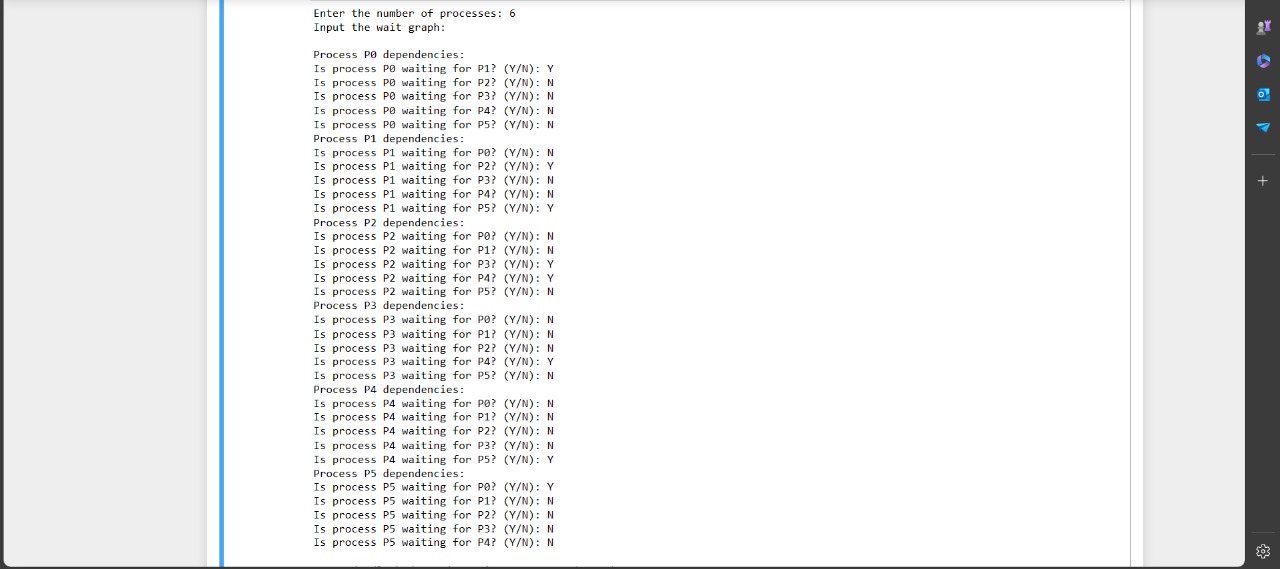
else:

print("\*")

print("\* NO DEADLOCK PRESENT \*")

print("\*")

**INPUT:**



**OUTPUT:**

Input deadlock detection trigger process (0 - 5): 0

====================================================

Wait-for Graph Process Dependencies:

P0 -> P1

P1 -> P2 P5

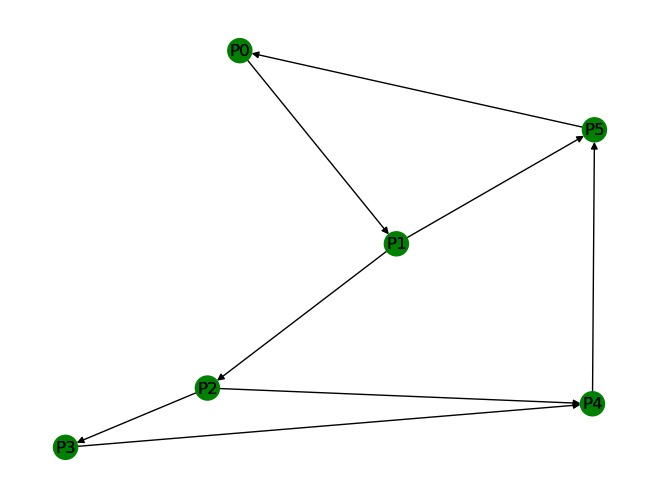
P2 -> P3 P4

P3 -> P4

P4 -> P5

P5 -> P0

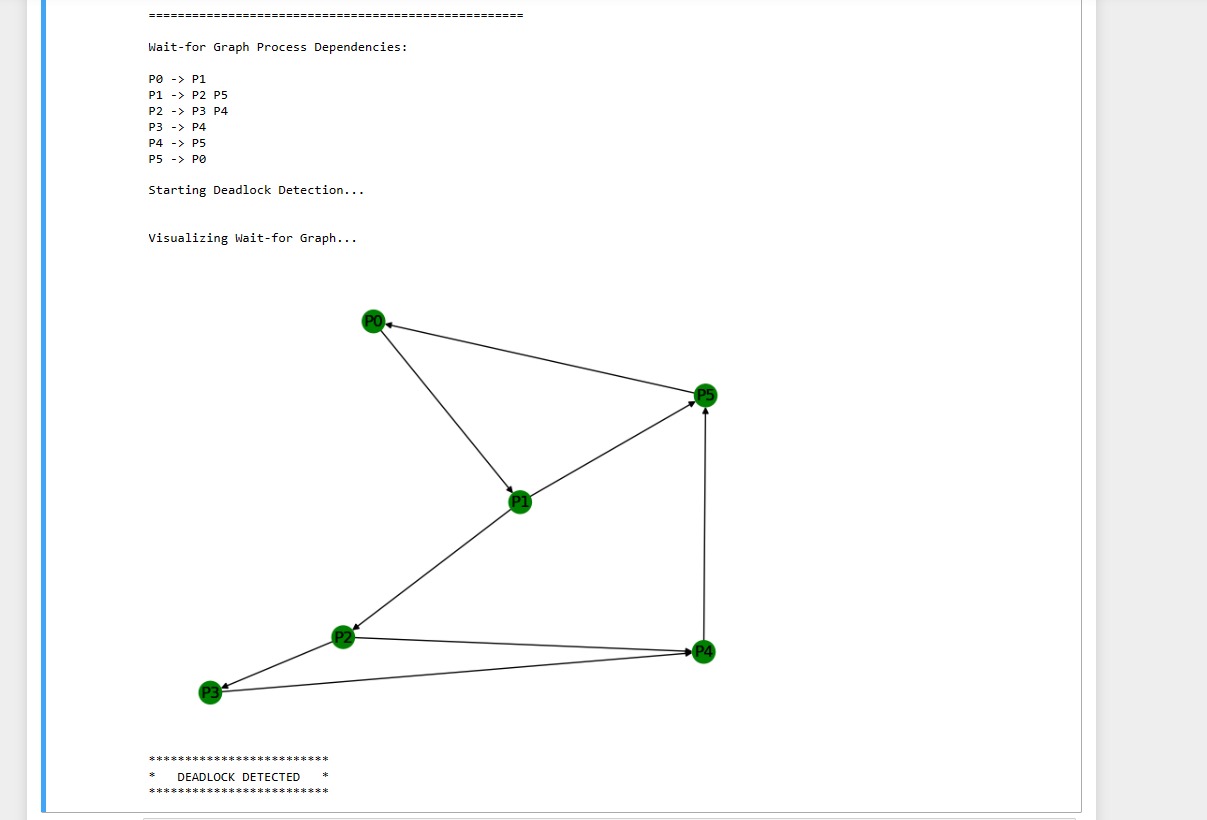
Starting Deadlock Detection...



Visualizing Wait-for Graph...

"\* DEADLOCK DETECTED \*"

**SCREENSHOT OF THE OUTPUT**



1. **ANALYSIS**

CODE:

import matplotlib.pyplot as plt

class Proc:

def \_init\_(self):

self.depProc = []

def visualize\_dependencies(processes, plot\_type='bar'):

num\_processes = len(processes)

process\_indices = list(range(num\_processes))

dependencies = [len(proc.depProc) for proc in processes]

if plot\_type == 'bar':

plt.bar(process\_indices, dependencies, color='blue')

elif plot\_type == 'line':

plt.plot(process\_indices, dependencies, marker='o', color='green', linestyle='-')

elif plot\_type == 'dot':

plt.scatter(process\_indices, dependencies, color='red')

elif plot\_type == 'hist':

plt.hist(dependencies, bins=range(max(dependencies)+2), color='orange', align='left')

plt.xlabel('Process')

plt.ylabel('Number of Dependencies')

plt.title(f'Dependencies of Each Process ({plot\_type.capitalize()} Plot)')

plt.xticks(process\_indices, [f'P{i}' for i in range(num\_processes)])

plt.grid(True)

plt.show()

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

nProcs = int(input("Enter the number of processes: "))

if nProcs > 1:

process = [Proc() for \_ in range(nProcs)]

print("Input the wait graph:\n")

for i in range(nProcs):

print(f"Process P{i} dependencies:")

for j in range(nProcs):

if i == j:

continue

validInput = False

while not validInput:

tmp = input(f"Is process P{i} waiting for P{j}? (Y/N): ")

if tmp.lower() == 'y':

validInput = True

process[i].depProc.append(j)

elif tmp.lower() == 'n':

validInput = True

else:

print("Invalid input. Please select either (Y/N)")

print("\n====================================================\n")

print("Wait-for Graph Process Dependencies:\n")

for i, proc in enumerate(process):

print(f"P{i} -> {' '.join(f'P{dep}' for dep in proc.depProc)}")

print("\nVisualizing Dependencies for Each Process...\n")

# Visualize using different plot types

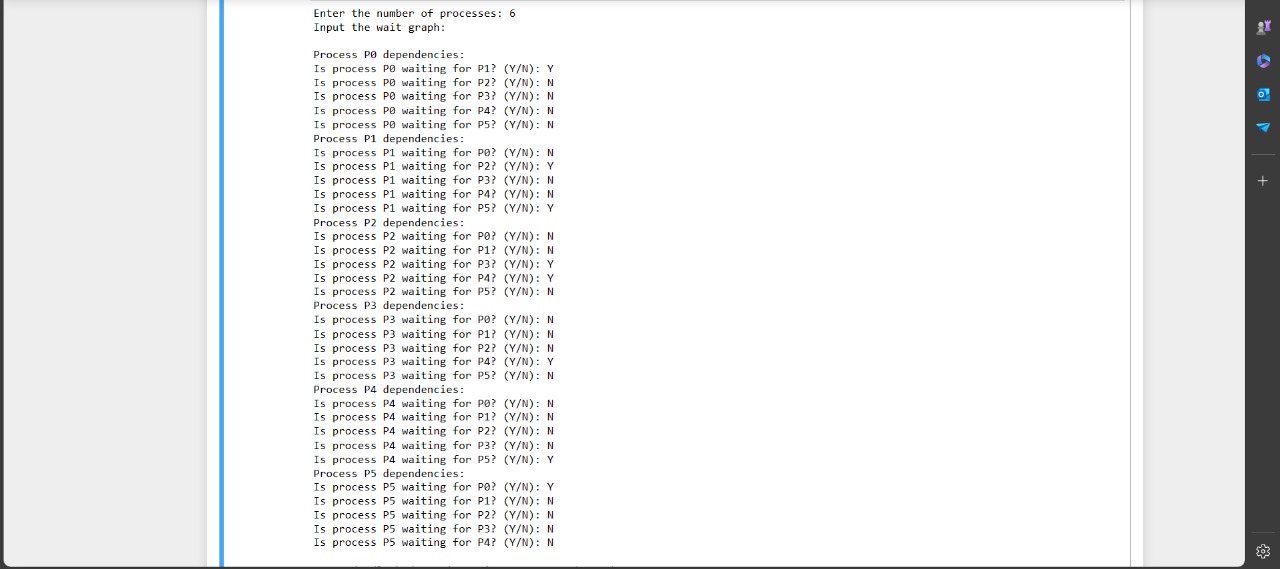
visualize\_dependencies(process, plot\_type='bar') # Bar plot

visualize\_dependencies(process, plot\_type='line') # Line plot

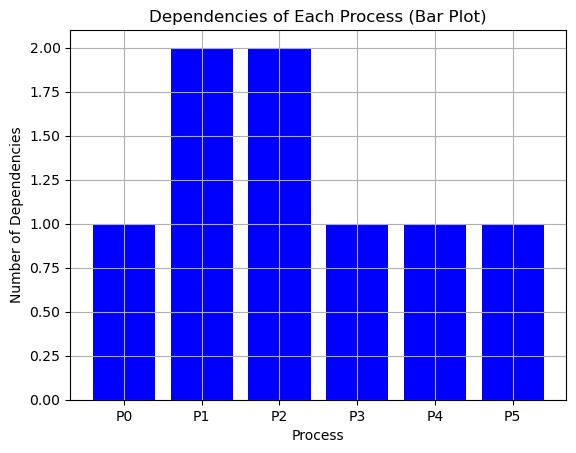
visualize\_dependencies(process, plot\_type='dot') # Dot plot

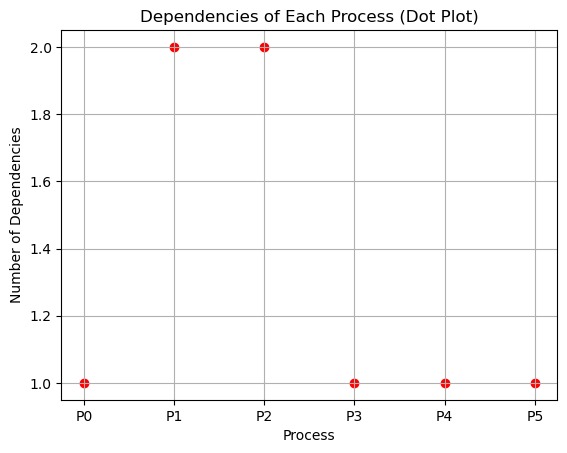
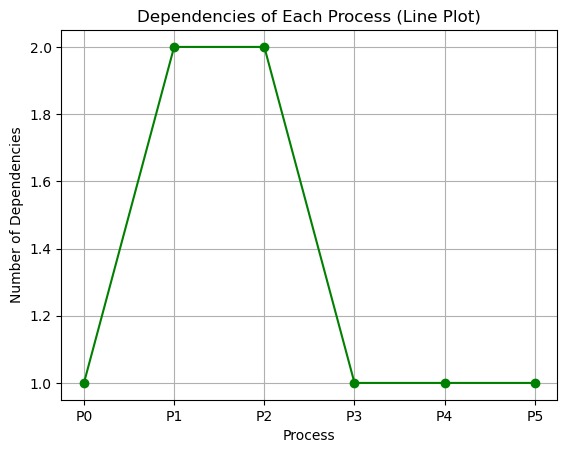
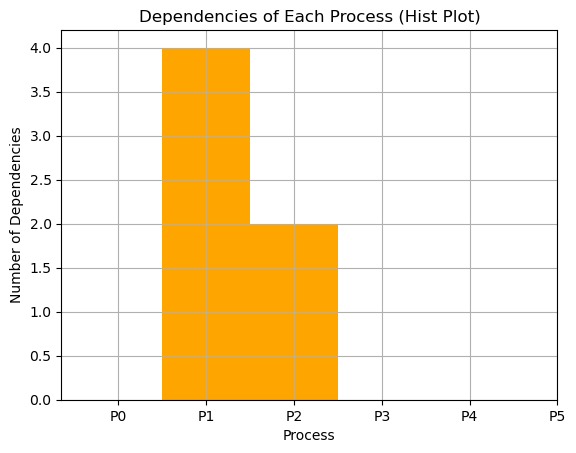
visualize\_dependencies(process, plot\_type='hist') # Histogram

INPUT:



OUTPUT:



**3. REPORT**

**1. Introduction:**

The purpose of this report is to analyze the deadlock detection using Chandy Misra Hass AND Model Algorithm and visualization process implemented in a given Python script. The analysis includes an overview of the input provided, the output generated, and insights into the deadlock detection algorithm and wait-for graph visualization.

**2. Input:**

* **Number of Processes**: 6
* **Wait Graph Dependencies**:
  + P0 depends on P1
  + P1 depends on P2 and P5
  + P2 depends on P3 and P4
  + P3 depends on P4
  + P4 depends on P5
  + P5 depends on P0
* **Deadlock Detection Trigger Process**: P0

**3. Output:**

* **Deadlock Detected**

**4. Analysis:**

* **Wait-for Graph Process Dependencies**:
  + The dependencies between processes are visualized, showing which processes are waiting for others.
* **Deadlock Detection**:
  + The deadlock detection algorithm explores the wait-for graph starting from the trigger process (P0) to identify cyclic dependencies.
  + Upon encountering a visited process during exploration, the algorithm flags a deadlock.
* **Visualization**:
  + The wait-for graph is visualized using NetworkX and Matplotlib, with nodes representing processes and edges representing dependencies.
  + Waiting processes are colored red, while non-waiting processes are green, aiding in identifying potential deadlocks.

**5. Conclusion:**

* The deadlock detection mechanism correctly identified a deadlock in the system.
* The wait-for graph visualization provides a clear representation of process dependencies, aiding in deadlock analysis and prevention.
* This analysis underscores the importance of implementing robust deadlock detection mechanisms in systems to maintain stability and reliability.