#!/bin/bash

# Create Project Directory

mkdir -p DeadlockDetectionSystem

cd DeadlockDetectionSystem

# Create Virtual Environment

python3 -m venv venv

source venv/bin/activate

# Ensure required packages are installed

pip install networkx matplotlib

# Create Project Structure

mkdir -p src

mkdir -p output

# Create requirements.txt

cat << EOF > requirements.txt

networkx==3.1

matplotlib==3.7.1

tkinter

EOF

# Create main.py

cat << 'EOF' > main.py

import os

import random

import tkinter as tk

from tkinter import ttk, messagebox

import networkx as nx

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

class ResourceAllocationGraph:

def \_\_init\_\_(self):

self.graph = nx.DiGraph()

plt.switch\_backend('Agg')

def add\_process(self, process\_id):

self.graph.add\_node(f'P{process\_id}', type='process')

def add\_resource(self, resource\_id, total\_instances):

self.graph.add\_node(f'R{resource\_id}', type='resource', instances=total\_instances)

def request\_resource(self, process\_id, resource\_id):

self.graph.add\_edge(f'P{process\_id}', f'R{resource\_id}', type='request')

def allocate\_resource(self, process\_id, resource\_id):

self.graph.add\_edge(f'R{resource\_id}', f'P{process\_id}', type='allocation')

def create\_graph\_visualization(self):

plt.figure(figsize=(8, 6))

pos = nx.spring\_layout(self.graph)

process\_nodes = [node for node in self.graph.nodes()

if self.graph.nodes[node]['type'] == 'process']

resource\_nodes = [node for node in self.graph.nodes()

if self.graph.nodes[node]['type'] == 'resource']

nx.draw\_networkx\_nodes(self.graph, pos,

nodelist=process\_nodes,

node\_color='lightblue',

node\_size=500,

label='Processes')

nx.draw\_networkx\_nodes(self.graph, pos,

nodelist=resource\_nodes,

node\_color='lightgreen',

node\_size=500,

label='Resources')

nx.draw\_networkx\_edges(self.graph, pos)

nx.draw\_networkx\_labels(self.graph, pos)

plt.title("Resource Allocation Graph")

plt.legend()

plt.axis('off')

return plt.gcf()

class DeadlockDetector:

def \_\_init\_\_(self, resource\_graph):

self.graph = resource\_graph.graph

def detect\_deadlock(self):

try:

cycles = list(nx.simple\_cycles(self.graph))

return {

'status': 'DEADLOCK' if cycles else 'NO\_DEADLOCK',

'cycles': cycles

}

except Exception as e:

return {

'status': 'ERROR',

'message': str(e)

}

def suggest\_resolution(self, deadlock\_info):

if deadlock\_info['status'] == 'DEADLOCK':

resolution\_steps = [

"1. Identify processes in the deadlock cycle",

"2. Choose a victim process to terminate",

"3. Release all resources held by the victim process",

"4. Allow other processes to continue execution"

]

return {

'steps': resolution\_steps,

'affected\_processes': [node for cycle in deadlock\_info['cycles'] for node in cycle if node.startswith('P')]

}

return {'steps': [], 'affected\_processes': []}

class DeadlockSimulation:

def \_\_init\_\_(self, num\_processes=5, num\_resources=3):

self.graph = ResourceAllocationGraph()

self.num\_processes = num\_processes

self.num\_resources = num\_resources

for i in range(num\_processes):

self.graph.add\_process(i)

for i in range(num\_resources):

self.graph.add\_resource(i, random.randint(1, 3))

def run\_simulation(self):

results = []

for \_ in range(10):

process = random.randint(0, self.num\_processes - 1)

resource = random.randint(0, self.num\_resources - 1)

if random.random() < 0.5:

self.graph.request\_resource(process, resource)

results.append(f"Process {process} requested Resource {resource}")

else:

self.graph.allocate\_resource(process, resource)

results.append(f"Process {process} allocated Resource {resource}")

detector = DeadlockDetector(self.graph)

deadlock\_result = detector.detect\_deadlock()

return {

'simulation\_log': results,

'deadlock\_result': deadlock\_result,

'resolution': detector.suggest\_resolution(deadlock\_result) if deadlock\_result['status'] == 'DEADLOCK' else None

}

class DeadlockDetectionApp:

def \_\_init\_\_(self, master):

self.master = master

master.title("Deadlock Detection Simulator")

master.geometry("800x600")

# Simulation Controls

control\_frame = ttk.Frame(master)

control\_frame.pack(pady=10)

ttk.Label(control\_frame, text="Processes:").grid(row=0, column=0, padx=5)

self.processes\_entry = ttk.Entry(control\_frame, width=10)

self.processes\_entry.grid(row=0, column=1, padx=5)

self.processes\_entry.insert(0, "5")

ttk.Label(control\_frame, text="Resources:").grid(row=0, column=2, padx=5)

self.resources\_entry = ttk.Entry(control\_frame, width=10)

self.resources\_entry.grid(row=0, column=3, padx=5)

self.resources\_entry.insert(0, "3")

run\_button = ttk.Button(control\_frame, text="Run Simulation", command=self.run\_simulation)

run\_button.grid(row=0, column=4, padx=10)

# Simulation Log

log\_frame = ttk.LabelFrame(master, text="Simulation Log")

log\_frame.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)

self.log\_text = tk.Text(log\_frame, wrap=tk.WORD, height=10)

self.log\_text.pack(padx=5, pady=5, fill=tk.BOTH, expand=True)

# Graph Visualization Frame

graph\_frame = ttk.LabelFrame(master, text="Resource Allocation Graph")

graph\_frame.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)

self.graph\_frame = graph\_frame

def run\_simulation(self):

# Clear previous results

for widget in self.graph\_frame.winfo\_children():

widget.destroy()

self.log\_text.delete(1.0, tk.END)

try:

num\_processes = int(self.processes\_entry.get())

num\_resources = int(self.resources\_entry.get())

except ValueError:

messagebox.showerror("Invalid Input", "Please enter valid numbers for processes and resources.")

return

# Run Simulation

simulation = DeadlockSimulation(num\_processes, num\_resources)

result = simulation.run\_simulation()

# Update Log

self.log\_text.insert(tk.END, "Simulation Log:\n")

for log\_entry in result['simulation\_log']:

self.log\_text.insert(tk.END, f"{log\_entry}\n")

self.log\_text.insert(tk.END, f"\nDeadlock Detection Result: {result['deadlock\_result']['status']}\n")

# Show Deadlock Resolution if Applicable

if result['deadlock\_result']['status'] == 'DEADLOCK':

self.log\_text.insert(tk.END, "\nDeadlock Resolution Steps:\n")

for step in result['resolution']['steps']:

self.log\_text.insert(tk.END, f"{step}\n")

self.log\_text.insert(tk.END, f"\nAffected Processes: {result['resolution']['affected\_processes']}\n")

# Visualize Graph

fig = simulation.graph.create\_graph\_visualization()

canvas = FigureCanvasTkAgg(fig, master=self.graph\_frame)

canvas\_widget = canvas.get\_tk\_widget()

canvas\_widget.pack(fill=tk.BOTH, expand=True)

canvas.draw()

def main():

root = tk.Tk()

app = DeadlockDetectionApp(root)

root.mainloop()

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

main()

EOF

# Create project files in src directory

cat << 'EOF' > src/\_\_init\_\_.py

from .resource\_graph import ResourceAllocationGraph

from .deadlock\_detector import DeadlockDetector

from .simulation import DeadlockSimulation

\_\_version\_\_ = "1.0.0"

\_\_authors\_\_ = ["Your Name"]

\_\_all\_\_ = [

'ResourceAllocationGraph',

'DeadlockDetector',

'DeadlockSimulation'

]

EOF

cat << 'EOF' > src/resource\_graph.py

import networkx as nx

import matplotlib.pyplot as plt

import os

class ResourceAllocationGraph:

def \_\_init\_\_(self):

self.graph = nx.DiGraph()

plt.switch\_backend('Agg')

def add\_process(self, process\_id):

self.graph.add\_node(f'P{process\_id}', type='process')

def add\_resource(self, resource\_id, total\_instances):

self.graph.add\_node(f'R{resource\_id}', type='resource', instances=total\_instances)

def request\_resource(self, process\_id, resource\_id):

self.graph.add\_edge(f'P{process\_id}', f'R{resource\_id}', type='request')

def allocate\_resource(self, process\_id, resource\_id):

self.graph.add\_edge(f'R{resource\_id}', f'P{process\_id}', type='allocation')

def visualize(self, output\_path=None):

plt.figure(figsize=(10, 6))

pos = nx.spring\_layout(self.graph)

process\_nodes = [node for node in self.graph.nodes()

if self.graph.nodes[node]['type'] == 'process']

resource\_nodes = [node for node in self.graph.nodes()

if self.graph.nodes[node]['type'] == 'resource']

nx.draw\_networkx\_nodes(self.graph, pos,

nodelist=process\_nodes,

node\_color='lightblue',

node\_size=500)

nx.draw\_networkx\_nodes(self.graph, pos,

nodelist=resource\_nodes,

node\_color='lightgreen',

node\_size=500)

nx.draw\_networkx\_edges(self.graph, pos)

nx.draw\_networkx\_labels(self.graph, pos)

plt.title("Resource Allocation Graph")

plt.axis('off')

if output\_path:

os.makedirs(os.path.dirname(output\_path), exist\_ok=True)

plt.savefig(output\_path)

plt.close()

else:

plt.show()

EOF

cat << 'EOF' > src/deadlock\_detector.py

import networkx as nx

class DeadlockDetector:

def \_\_init\_\_(self, resource\_graph):

self.graph = resource\_graph.graph

def detect\_deadlock(self):

try:

cycles = list(nx.simple\_cycles(self.graph))

return {

'status': 'DEADLOCK' if cycles else 'NO\_DEADLOCK',

'cycles': cycles

}

except Exception as e:

return {

'status': 'ERROR',

'message': str(e)

}

def suggest\_resolution(self, deadlock\_info):

if deadlock\_info['status'] == 'DEADLOCK':

resolution\_steps = [

"1. Identify processes in the deadlock cycle",

"2. Choose a victim process to terminate",

"3. Release all resources held by the victim process",

"4. Allow other processes to continue execution"

]

return {

'steps': resolution\_steps,

'affected\_processes': [node for cycle in deadlock\_info['cycles'] for node in cycle if node.startswith('P')]

}

return {'steps': [], 'affected\_processes': []}

EOF

cat << 'EOF' > src/simulation.py

import random

from .resource\_graph import ResourceAllocationGraph

from .deadlock\_detector import DeadlockDetector

class DeadlockSimulation:

def \_\_init\_\_(self, num\_processes=5, num\_resources=3):

self.graph = ResourceAllocationGraph()

self.num\_processes = num\_processes

self.num\_resources = num\_resources

for i in range(num\_processes):

self.graph.add\_process(i)

for i in range(num\_resources):

self.graph.add\_resource(i, random.randint(1, 3))

def run\_simulation(self):

results = []

for \_ in range(10):

process = random.randint(0, self.num\_processes - 1)

resource = random.randint(0, self.num\_resources - 1)

if random.random() < 0.5:

self.graph.request\_resource(process, resource)

results.append(f"Process {process} requested Resource {resource}")

else:

self.graph.allocate\_resource(process, resource)

results.append(f"Process {process} allocated Resource {resource}")

detector = DeadlockDetector(self.graph)

deadlock\_result = detector.detect\_deadlock()

return {

'simulation\_log': results,

'deadlock\_result': deadlock\_result,

'resolution': detector.suggest\_resolution(deadlock\_result) if deadlock\_result['status'] == 'DEADLOCK' else None

}

EOF

# Install system dependencies if not already installed

sudo apt-get update

sudo apt-get install -y python3-tk python3-matplotlib python3-networkx

# Install Python dependencies

pip install -r requirements.txt

# Print completion message

echo "Deadlock Detection System project created successfully!"

echo "To run the application, activate the virtual environment and run:"

echo "python main.py"

EOF

Now, follow these steps:

1. Copy the entire script

2. Open your terminal

3. Paste the script and press Enter

4. It will create the project and install dependencies

5. When done, navigate into the project:

```

cd DeadlockDetectionSystem

source venv/bin/activate

python main.py

```

Key things to note:

- You need `sudo` permissions to install system dependencies

- Ensure you have Python 3 and pip installed

- The script will create a virtual environment and install all necessary packages

Would you like me to clarify anything about the script?