

**CSE316-OPERATING SYSTEMS**

**K23CT**

Name: B Poorvi Shasta

Reg No: 12301224

Roll No: 04

Name : Jayavarapu Mohan Abhishek Gupta

Reg No: 12306984

Roll No: 45

Name: Kusmitha

Reg No: 12315122

Roll No: 48

Submitted to: Gangandeep Kour

**Automated Deadlock Detection Tool Report**

**1.Project Overview**

**Project Title:** Automated Deadlock Detection Tool

**Goal:**

* Develop a tool that automatically detects potential deadlocks in system processes.
* Analyze process dependencies and resource allocations to identify circular wait conditions.
* Suggest resolution strategies.

**Expected Outcomes:**

* Real-time deadlock detection and visualization.
* Resolution suggestions to prevent system hangs.
* User-friendly interface for process monitoring.

**Scope:**

* **Input:** System process and resource allocation details.
* **Output:** Deadlock detection, visualization, and resolution suggestions.
* **Constraints:** Real-time detection with minimal overhead.

**2. Module-Wise Breakdown**

**Module 1: Process and Resource Analyzer (Backend)**

* **Purpose:**
  + Extract system processes and resource allocations.
  + Map process dependencies to construct a **Wait-for Graph**.
* **Functionalities:**
  + Use Python’s psutil library to access process and resource details.
  + Store process dependencies in a graph structure.
* **Example:**
  + List running processes and allocated resources.
  + Display process-resource mapping.

**Module 2: Deadlock Detection Engine (Backend)**

* **Purpose:**
  + Detect circular wait conditions using graph algorithms.
* **Functionalities:**
  + Use **DFS (Depth-First Search)** to detect cycles in the **Wait-for Graph**.
  + Suggest resolution strategies (resource preemption or process termination).
* **Example:**
  + Identify circular dependencies like P1 → P2 → P3 → P1.
  + Suggest terminating P2 or preempting its resource.

**Module 3: Visualization & Reporting (Frontend)**

* **Purpose:**
  + Graphically represent the process-resource relationship.
  + Display logs and generate reports.
* **Functionalities:**
  + Use **HTML/CSS/JavaScript** for the GUI.
  + Display live data with color-coded nodes (red = deadlocked, green = running).
  + Generate downloadable reports.
* **Example:**
  + Graph shows deadlocked processes in red and running processes in green.

**3. Functionalities :**

|  |  |  |
| --- | --- | --- |
| Module | Functionalities | Example |
| Process & Resource Analyzer | **Extract running processes and track resources.** | **Display process table.** |
|  | **Map process-resource dependencies.** | **Show allocation matrix.** |
| Deadlock Detection Engine | **Detect circular waits using DFS.** | **Identify deadlocks.** |
|  | **Suggest resolution strategies.** | **Preempt resource, terminate P2.** |
| Visualization & Reporting | **Display live processes and deadlocks visually.** | **Color-coded graph.** |
|  | **Generate logs and reports.** | **Downloadable PDF report.** |

**4. Technology Used**

**Programming Languages:**

* **Python:** Backend for process analysis and deadlock detection.
* **HTML/CSS/JavaScript:** Frontend GUI and visualization.

**Libraries and Tools:**

* **Python:**
  + psutil → To extract system processes and resources.
  + networkx → For graph representation and cycle detection.
  + flask → To create a web server.
* **Frontend (HTML/CSS/JS):**
  + D3.js → For dynamic graph visualization.
* **Other Tools:** GitHub for version control, Unit Testing Frameworks.

**5. Flow Diagram :**

**6. Revision Tracking on GitHub:**

* **Repository Name:** Automated-Deadlock-Detection-Tool
* **GitHub Link:** <https://github.com/PoorviShasta/Deadlock_detection_tool_OS>
* **Version Control:**
* **Main Branch:** Stable production code.
* **Feature Branches:** Individual modules.
* **Commits:**
  + commit 1: Project initialization.
  + commit 2: Backend process extraction.
  + commit 3: Deadlock detection logic.
  + commit 4: Frontend GUI implementation.
  + commit 5: Visualization and reporting.
  + commit 6: Code optimization and bug fixes.
  + commit 7: Final version with documentation.

**7. Conclusion and Future Scope:**

The **Automated Deadlock Detection Tool** offers a comprehensive solution for detecting deadlocks in process-resource systems. Its automatic detection and resolution suggestion capabilities make it suitable for enhancing process management in various environments.

**Future Scope:**

* Enhancing the tool to support **distributed systems**.
* Integrating **machine learning algorithms** for deadlock prediction and prevention.
* Allowing **real-time automatic resolution**.
* Improving the GUI for more detailed visual insights.

**8. References:**

* **Books:**
  + Silberschatz, Galvin, Gagne – Operating System Concepts.
* **Online Sources:**
  + Python psutil documentation: <https://psutil.readthedocs.io>
  + Flask documentation: https://flask.palletsprojects.com
  + GitHub Docs: <https://docs.github.com>

**Appendix**

A. **AI-Generated Project Elaboration/Breakdown Report:** (Include detailed breakdown here)

**Understanding the Project:**

This project is an Automated Deadlock Detection Tool that helps analyze process dependencies and resource allocations to identify potential deadlocks. It is built as a web-based tool where users input process and resource details, and the system determines whether a deadlock exists.

The detection is based on the Resource Allocation Graph (RAG) model and uses graph traversal techniques to check for circular dependencies. If a deadlock is found, the tool highlights the affected processes and provides resolution strategies.

**Working of the Tool:**

**1️. User Inputs Information**

Specifies the number of processes and number of resources.

Enters the allocation matrix (showing currently held resources).

Enters the request matrix (showing requested resources)

**2️. Backend Processing**

The tool processes the matrices and constructs a graph of dependencies.

The system checks for circular wait conditions using Graph traversal (DFS) and Banker’s Algorithm (if needed for safety checks)

**3️. Detecting Deadlock & Suggestions**

If a deadlock is found, it highlights the involved processes.

If no deadlock is found, it shows a safe execution order.

**Technologies Used:**

Frontend:HTML, CSS, JavaScript (For user interaction)

Backend: Python, Flask (For processing requests and detecting deadlocks)

B. **Problem Statement:** In an operating system, deadlocks occur when multiple processes wait indefinitely for resources held by each other. This can slow down or crash critical systems, leading to performance issues.

This project aims to automate deadlock detection using a web-based tool that takes real-time input from users and checks for circular dependencies in resource allocation.

C. **Solution/Code:** (Complete code to be added here)

1. **app.py** (Flask Backend)

from flask import Flask, render\_template, request, jsonify

import numpy as np

import deadlock

app = Flask(\_\_name\_\_)

@app.route('/')

def home():

return render\_template('index.html')

@app.route('/detect', methods=['POST'])

def detect():

data = request.json

processes = int(data['processes'])

resources = int(data['resources'])

allocation = np.array(data['allocation']).reshape(processes, resources)

request\_matrix = np.array(data['request']).reshape(processes, resources)

deadlock\_detected, message = deadlock.detect\_deadlock(processes, resources, allocation, request\_matrix)

return jsonify({"deadlock": deadlock\_detected, "message": message})

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

app.run(debug=True)

2. **deadlock.py** (Deadlock Detection Algorithm)

import numpy as np

def detect\_deadlock(processes, resources, allocation, request):

    work = np.zeros(resources)

    finish = [False] \* processes

    for j in range(resources):

        work[j] = sum(allocation[i][j] for i in range(processes)) - sum(request[i][j] for i in range(processes))

    safe\_sequence = []

    while len(safe\_sequence) < processes:

        allocated = False

        for i in range(processes):

            if not finish[i] and all(request[i][j] <= work[j] for j in range(resources)):

                safe\_sequence.append(i)

                work += allocation[i]

                finish[i] = True

                allocated = True

                break

        if not allocated:

            return True, "Deadlock detected!"

    return False, f"No deadlock detected. Safe sequence: {safe\_sequence}"

3. **index.html** (Frontend )

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Deadlock Detection Tool</title>

    <link rel="stylesheet" href="{{ url\_for('static', filename='style.css') }}">

</head>

<body>

    <h1>Automated Deadlock Detection Tool</h1>

    <form id="deadlockForm">

        <label>Number of Processes:</label>

        <input type="number" id="processes" min="1" required><br><br>

        <label>Number of Resources:</label>

        <input type="number" id="resources" min="1" required><br><br>

        <label>Allocation Matrix (comma-separated, row by row):</label>

        <textarea id="allocation" rows="4" placeholder="e.g., 0,1,0, 2,0,0, 0,0,1" required></textarea><br><br>

        <label>Request Matrix (comma-separated, row by row):</label>

        <textarea id="request" rows="4" placeholder="e.g., 0,0,1, 0,1,0, 1,0,0" required></textarea><br><br>

        <button type="submit">Detect Deadlock</button>

    </form>

    <h2>Result:</h2>

    <p id="result"></p>

    <script src="{{ url\_for('static', filename='script.js') }}"></script>

</body>

</html>

4. **style.css** (Frontend Styling)

body {

    font-family: 'Arial', sans-serif;

    text-align: center;

    background: linear-gradient(to right, #f8f9fa, #e9ecef);

    margin: 0;

    padding: 20px;

}

.container {

    width: 50%;

    margin: auto;

    background: white;

    padding: 25px;

    border-radius: 10px;

    box-shadow: 0px 5px 15px rgba(0, 0, 0, 0.2);

    transition: 0.3s;

}

.container:hover {

    transform: scale(1.02);

}

input, textarea {

    display: block;

    width: 90%;

    margin: 10px auto;

    padding: 12px;

    border: 2px solid #ccc;

    border-radius: 8px;

    font-size: 16px;

    transition: 0.3s;

}

input:focus, textarea:focus {

    border-color: #007BFF;

    outline: none;

    box-shadow: 0px 0px 5px rgba(0, 123, 255, 0.5);

}

button {

    padding: 14px 22px;

    background: #007BFF;

    color: white;

    font-size: 18px;

    border: none;

    border-radius: 8px;

    cursor: pointer;

    transition: 0.3s;

}

button:hover {

    background: #0056b3;

    transform: scale(1.05);

}

.result-box {

    margin-top: 20px;

    padding: 20px;

    font-size: 18px;

    font-weight: bold;

    display: none;

    border-radius: 8px;

}

.success {

    background: #d4edda;

    border-left: 6px solid #28a745;

    color: #155724;

}

.error {

    background: #f8d7da;

    border-left: 6px solid #dc3545;

    color: #721c24;

}

5. **script.js** (JavaScript Logic)

document.getElementById('deadlockForm').addEventListener('submit', async (e) => {

    e.preventDefault();

    const processes = document.getElementById('processes').value;

    const resources = document.getElementById('resources').value;

    const allocationMatrix = document.getElementById('allocation').value

        .trim().split(/\s\*[,|\s]\s\*/).map(Number);

    const requestMatrix = document.getElementById('request').value

        .trim().split(/\s\*[,|\s]\s\*/).map(Number);

    const allocation = [];

    const request = [];

    for (let i = 0; i < processes; i++) {

        allocation.push(allocationMatrix.slice(i \* resources, (i + 1) \* resources));

        request.push(requestMatrix.slice(i \* resources, (i + 1) \* resources));

    }

    const response = await fetch('/detect', {

        method: 'POST',

        headers: { 'Content-Type': 'application/json' },

        body: JSON.stringify({ processes, resources, allocation, request })

    });

    const result = await response.json();

    const resultBox = document.getElementById('result-box');

    document.getElementById('result').innerText = result.message;

    resultBox.style.display = "block";

    if (result.deadlock) {

        resultBox.className = "result-box error";

    } else {

        resultBox.className = "result-box success";

    }

});