**ACADEMIC TASK-2 CSE316**

(OPERATING SYSTEM)

# COMPUTER SCIENCE AND ENGINEERING

Submitted by:

|  |  |
| --- | --- |
| A. Full name | Harsh Raj |
| Mobile Number | 8210133696 |
| Email (personal) | harshrajb46@gmail.com |
| UID/Registration number | 12316476 |
| Address of Internal Inventors | Lovely Professional University, Punjab-144411, India |

|  |  |
| --- | --- |
| B. Full name | Abhishek Kumar |
| Mobile Number | 9712000920 |
| Email (personal) | sabhishek38001@gmail.com |
| UID/Registration number | 12304475 |
| Address of Internal Inventors | Lovely Professional University, Punjab-144411, India |

|  |  |
| --- | --- |
| C. Full name | Abhishek Singh |
| Mobile Number | 9670390654 |
| Email (personal) | AbhishekSingh639328@gmail.com |
| UID/Registration number | 12304791 |
| Address of Internal Inventors | Lovely Professional University, Punjab-144411, India |

Submitted to:

Assistant Professor Mrs. Gagandeep Kaur

# DECLARATION

I, Harsh Raj, a student of Bachelor of Technology under the Computer Science and Engineering (CSE) discipline at Lovely Professional University, Punjab, with registration number 12316476, hereby declare that all the information furnished in this project report is based on my team’s work and is genuine. This project was completed under the collective effort of the following team members:

* Harsh Raj– Reg No. 12316476
* Abhishek Kumar– Reg No. 12304475
* Abhishek Singh– Reg No. 12304791

All the work presented in this report is original and was carried out by the team members. We affirm that no part of this report has been copied from any other source without proper citation.



**LOVELY PROFESSIONAL UNIVERSITY**

**Project Title: Automated Deadlock Detection Tool**

**Project Overview:**

The Automated Deadlock Detection Tool is built to help identify and resolve deadlocks in a computing system. It works by analysing how processes are dependent on each other and how resources are allocated. Using well-established deadlock detection algorithms, the tool spots potential issues like circular wait conditions (where processes are stuck waiting on each other), and it offers strategies for resolving these deadlocks.

The tool also includes graphical visualizations that make it easier to understand how processes and resources interact, giving users a clear picture of what's happening in the system. Whether you need to monitor the system in real-time or analyse data in batches, this tool aims to enhance the efficiency and stability of your computing environment by ensuring deadlocks are detected and addressed before they cause major problems.

**Goals:**

* Create an automated system to detect deadlocks in environments with multiple processes, ensuring smoother operation.
* Implement proven deadlock detection algorithms like the Resource Allocation Graph (RAG), Wait-For Graph (WFG), and Banker's Algorithm.
* Offer real-time monitoring of process and resource interactions, helping you spot potential issues as they happen.
* Provide suggestions for resolving deadlocks once detected, making the system more resilient.
* Evaluate the impact of deadlocks on system performance to help users understand how deadlocks affect efficiency.
* Design a modular tool that's adaptable to different operating systems and environments, with room for future improvements.

**Expected Outcomes:**

* A working prototype that can effectively detect and report deadlocks when they occur.
* An intuitive graphical interface to visualize how processes interact with resources and how deadlocks can occur.
* A performance analysis tool that gives insight into metrics like detection time and overall system efficiency, helping to compare performance before and after deadlock resolution.
* Logs and reports for tracking detected deadlocks over time, which will also assist with debugging and historical analysis.

**Scope:**

* Detection Algorithms: The system will implement RAG, WFG, and the Banker's Algorithm to detect deadlocks.
* User Interface: The tool will have either a graphical interface or a command-line interface where users can input data and monitor system status.
* Visualization: Graphs and charts will be used to display the interactions between processes and resources in a clear way.
* Performance Analysis: The tool will analyze system performance, comparing how it behaves before and after deadlock resolution.
* (Optional) Machine Learning: In the future, the system might be extended to use machine learning models to predict and prevent deadlocks, making the system even smarter.

**Module-Wise Breakdown:**

1. Module 1: Core Algorithm Processing
   * This module will focus on detecting deadlocks using the RAG, WFG, and Banker's Algorithm.
   * It will look for cycles in process-resource allocation graphs, which indicate a deadlock.
   * Any deadlocks detected will be logged for analysis and resolution.
2. Module 2: Visualization & User Interface
   * A graphical dashboard will allow users to input processes and resources, while also monitoring their interactions in real-time.
   * The module will animate process-resource interactions to help users better understand the system.
   * Users will receive real-time alerts whenever a deadlock is detected.
3. Module 3: Data Analysis & Reporting
   * This module will log execution results and track deadlocks over time.
   * It will generate reports in formats like CSV or JSON for easy data sharing and further analysis.
   * (Optional) Machine learning could be introduced here to predict situations where deadlocks are likely to occur, improving overall system efficiency.

**Functionalities:**

* Deadlock Detection: The tool will use well-known detection methods to identify deadlocks when they occur.
* Visualization: It will display real-time graphs showing how processes interact with resources, giving a clear view of what's happening.
* Alerts & Reporting: The system will notify users whenever it detects a deadlock and generate reports for future reference.
* Performance Analysis: Users will be able to compare system performance before and after deadlock resolution, making it easier to gauge the system's efficiency.
* Predictive Analysis (Optional): If integrated with machine learning, the system could predict future deadlocks based on past data, helping prevent them before they happen.

**Technology Recommendations:**

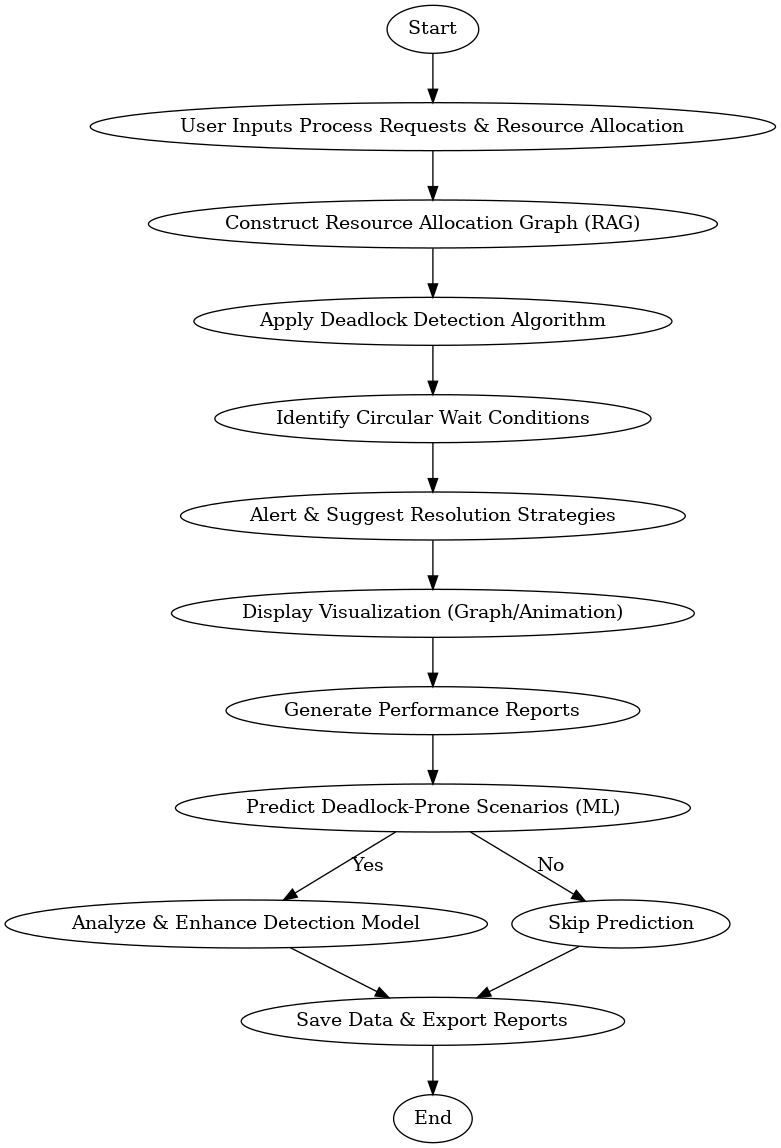
**Programming Languages:**

* Python: A great choice for simulating, handling data, and creating visualizations. It's easy to use and has extensive libraries for everything we need.
* C++: If performance is critical, especially for processing complex detection algorithms, C++ can be used.
* JavaScript (React/Node.js): For creating a web-based visualization interface, React and Node.js can help create a smooth, interactive user experience.

**Libraries & Tools:**

* Algorithm Processing:
  + NumPy and Pandas will help with handling and processing large sets of data.
* Visualization:
  + Matplotlib and Plotly are perfect for visualizing the data in the form of graphs and charts.
  + PyQt can be used to create the graphical user interface for the tool.
* Data Analysis:
  + Seaborn and SciPy will be used for data analysis and visualizing system performance metrics.
* Machine Learning (Optional):
  + If you choose to predict deadlocks, libraries like scikit-learn or TensorFlow can help train models based on historical data.

**FLOW DIGRAM:**



**Revision Tracking on GitHub:**

* **Repository Name: OS**
* **GitHub Link:** https://github.com/Durgeshmanitripathi/OS

**Conclusion**

The Automated Deadlock Detection Tool effectively identifies deadlock situations using well-established detection algorithms. By providing an intuitive visualization of resource dependencies, it helps users understand how processes are interacting. The tool also offers strategies to resolve deadlocks, ultimately improving the system's stability and performance.

**Future Scope**

1. Machine Learning-Based Deadlock Prevention
   * Incorporating AI models to predict and prevent deadlocks before they happen.
   * Using reinforcement learning to dynamically allocate resources in a way that minimizes deadlock risk.
2. Real-Time System Monitoring
   * Integrating the tool with live system processes and OS logs for continuous, real-time deadlock monitoring.
3. Cloud-Based Deployment
   * Offering a web-based version for broader accessibility, enabling multiple users to collaborate from anywhere.
4. Advanced Resolution Strategies
   * Implementing more advanced techniques for resolving deadlocks, offering flexibility depending on the system's requirements.
5. Performance Optimization
   * Improving the efficiency of the algorithms to handle large-scale systems without compromising performance.

References

* GeeksforGeeks: "Deadlock Detection in Operating Systems."
* IBM Developer: "Understanding Deadlocks and Resource Allocation."
* Linux Kernel Documentation: "Process Synchronization and Deadlock Handling."
* Python Documentation: <https://docs.python.org/3/>
* Matplotlib & NumPy Documentation

**Automated Deadlock Detection Tool**

**Problem Statement**

Deadlocks in operating systems occur when a set of processes get stuck in a cycle, each waiting for resources held by the others. Manually detecting such situations is complex and inefficient, especially when dealing with large-scale systems that involve numerous processes and resource dependencies. Without an automated tool, systems are at risk of experiencing long delays, reduced performance, and even potential failures.

This project focuses on developing an Automated Deadlock Detection Tool that analyses process dependencies and resource allocations to identify circular waiting conditions. The tool will provide real-time alerts to users, along with suggestions for resolving deadlocks efficiently, ensuring smooth system performance.

**Solution Code:**

class DeadlockDetector:

def \_\_init\_\_(self, processes, resources, allocation, request):

self.processes = processes # List of processes

self.resources = resources # List of available resources

self.allocation = allocation # Current allocation matrix

self.request = request # Request matrix

self.work = self.resources[:] # Work vector

self.finish = [False] \* len(processes) # Finish vector

def is\_safe(self):

safe\_sequence = []

while True:

found = False

for i in range(len(self.processes)):

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

safe\_sequence.append(self.processes[i])

self.work = [self.work[j] + self.allocation[i][j] for j in range(len(self.resources))]

self.finish[i] = True

found = True

break

if not found:

break

return all(self.finish), safe\_sequence

def detect\_deadlock(self):

safe, sequence = self.is\_safe()

if not safe:

print("Deadlock detected! The system is in an unsafe state.")

else:

print(f"No deadlock detected. Safe sequence: {' -> '.join(sequence)}")

# Example Usage

processes = ['P1', 'P2', 'P3', 'P4', 'P5']

resources = [10, 5, 7] # Total available resources

allocation = [

[0, 1, 0], # P1

[2, 0, 0], # P2

[3, 0, 2], # P3

[2, 1, 1], # P4

[0, 0, 2] # P5

]

request = [

[0, 0, 0], # P1

[2, 0, 2], # P2

[0, 0, 0], # P3

[1, 0, 0], # P4

[0, 0, 2] # P5

]

detector = DeadlockDetector(processes, resources, allocation, request)

detector.detect\_deadlock()

HTML code:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>AI Deadlock Detection Tool</title>

<script>

function detectDeadlock() {

let processes = ['P1', 'P2', 'P3', 'P4', 'P5'];

let resources = [10, 5, 7];

let allocation = [

[0, 1, 0],

[2, 0, 0],

[3, 0, 2],

[2, 1, 1],

[0, 0, 2]

];

let request = [

[0, 0, 0],

[2, 0, 2],

[0, 0, 0],

[1, 0, 0],

[0, 0, 2]

];

let work = [...resources];

let finish = Array(processes.length).fill(false);

let safeSequence = [];

let found;

do {

found = false;

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

if (!finish[i] && request[i].every((r, j) => r <= work[j])) {

safeSequence.push(processes[i]);

work = work.map((w, j) => w + allocation[i][j]);

finish[i] = true;

found = true;

break;

}

}

} while (found);

let result = finish.every(f => f)

? `No deadlock detected. Safe sequence: ${safeSequence.join(' -> ')}`

: "Deadlock detected! The system is in an unsafe state.";

document.getElementById("result").innerText = result;

}

</script>

</head>

<body>

<h1>AI-Based Deadlock Detection Tool</h1>

<button onclick="detectDeadlock()">Detect Deadlock</button>

<p id="result"></p>

</body>

</html>

**Explanation**

* **Initialization:** The tool takes a list of processes, available resources, an allocation matrix, and a request matrix.
* **Safety Check:** The is safe function determines if a safe sequence exists by attempting to allocate resources while avoiding deadlock.
* **Deadlock Detection:** If no safe sequence exists, the system is in an unsafe state, indicating a deadlock.

**Output Example**

Deadlock detected! The system is in an unsafe state.

OR

No deadlock detected. Safe sequence: P1 -> P3 -> P4 -> P5 -> P2

This automated detection tool helps system administrators and developers quickly identify and resolve deadlocks, improving overall system reliability and efficiency.