

**CSE316-OPERATING SYSTEMS**

**K23GN**

Name: SAHIL

Reg No: 12322937

Roll No: 14

Name: VISHAL SHUKLA

Reg No: 12306876

Roll No: 54

Name: MOHAMMED SANAD K C

Reg No: 12325793

Roll No: 20

Submitted to: Dr.Parvinder Singh

**Project Title:-**

**Deadlock Detection and Recovery**

**1. Goal:-**

The objective of this project is to develop a Graphical User Interface (GUI)-based system that allows users to:

* Detect deadlocks in process-resource systems.
* View the current state of processes and resources.
* Recover from deadlocks using two strategies:
  + Process Termination
  + Resource Preemption

The system enables users to manually manage deadlocks and enhances understanding of resource allocation issues in operating systems.

**2. System Design and Working:-**

**Key Components**

| Component | Description |
| --- | --- |
| Process Class | Represents an individual process with ID, maximum resources required, currently allocated resources, and remaining need. |
| DeadlockDetector Class | Contains logic for detecting deadlocks using Banker's Algorithm principles. |
| DeadlockApp Class | Builds the GUI using Tkinter, allows users to input processes/resources, and handles deadlock detection and recovery. |

**3. System Workflow:-**

1. Input Resources:
   * User enters the total number of available resources.
2. Input Processes:
   * For each process, the user enters:
     + Process ID.
     + Maximum resource requirements.
3. Deadlock Detection:
   * When the user clicks "Check Deadlock", the system:
     + Applies the Safe Sequence Check (similar to Banker's Algorithm).
     + If a safe sequence exists → No deadlock.
     + If not → Deadlock detected.
4. Deadlock Recovery:
   * If a deadlock is detected:
     + Users can terminate a process.
     + Or preempt resources from a process.

**4. Module-wise Breakdown:-**

**Process Class**

Purpose:

* + Store process ID, maximum resource needs, current allocations, and needs.
* Attributes:
  + pid: Process ID.
  + max\_resources: Maximum resources needed.
  + allocated: Resources currently allocated (initially zero).
  + need: Remaining resources needed (initially equal to max\_resources).

**Deadlock Detector Class:-**

* Purpose:
  + Detect deadlocks by checking if the system can allocate resources safely.
* Methods:
  + is\_safe():
    - Tries to allocate resources sequentially to processes.
    - If every process can finish → No deadlock.
    - Otherwise → Deadlock detected.
  + detect\_deadlock():
    - Returns whether a deadlock exists and the safe sequence if available.

DeadlockApp Class (Tkinter GUI)

* Purpose:
  + Provide a user-friendly interface to interact with the system.
* Features:
  + Input resource data.
  + Add processes.
  + Display system state.
  + Detect deadlocks and show recovery options:
    - Terminate a process.
    - Preempt resources from a process.
* Recovery Options:
  + Termination:
    - Remove a selected process from the system.
  + Preemption:
    - Selectively reclaim resources from a process to resolve deadlock.

**5. Technologies Used:-**

| Technology | Purpose |
| --- | --- |
| Python 3 | Backend logic and process management. |
| Tkinter | GUI development for user interactions. |
| Messagebox (tkinter.messagebox) | Display messages and alerts. |

**6. Flow Diagram:-**

pgsql

CopyEdit

+---------------------------------+

| User Inputs Resources & Processes |

+---------------------------------+

↓

+---------------------------------+

| Deadlock Detection Module |

| (Safe State Check / Cycle Detection) |

+---------------------------------+

↓

+------------------------+-------------------------+

| | |

No Deadlock Deadlock Detected --> Recovery Options Shown

| | |

Safe Sequence Terminate Process / Preempt Resources

↓ ↓

System Continues System Moves Toward Safe State

**7. Conclusion:-**

The Deadlock Detection and Recovery System developed using Tkinter successfully demonstrates:

* Detection of unsafe states (deadlocks) in a process-resource system.
* Manual recovery strategies like termination and resource preemption.
* Real-time visualization of system state and dynamic user interaction.

This system not only provides practical solutions to deadlock management but also serves as a valuable educational tool for understanding deadlock concepts in Operating Systems.

**8. Future Enhancements:-**

* Dynamic Resource Allocation:
  + Allow users to update resource allocations dynamically while the system runs.
* Priority-based Recovery:
  + Select the best process for termination or preemption based on priority levels.
* Deadlock Prediction:
  + Integrate predictive analytics to forecast possible future deadlocks.
* Distributed Deadlock Detection:
  + Expand the tool to simulate deadlock detection in distributed environments.

**9. CODE:-**

import tkinter as tk

from tkinter import messagebox

class Process:

def \_\_init\_\_(self, pid, max\_resources):

self.pid = pid

self.max\_resources = max\_resources

self.allocated = [0] \* len(max\_resources)

self.need = max\_resources[:]

class DeadlockDetector:

def \_\_init\_\_(self, processes, available\_resources):

self.processes = processes

self.available\_resources = available\_resources

def is\_safe(self):

work = self.available\_resources[:]

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

safe\_sequence = []

while len(safe\_sequence) < len(self.processes):

progress\_made = False

for i, process in enumerate(self.processes):

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

work = [work[j] + process.allocated[j] for j in range(len(work))]

finish[i] = True

safe\_sequence.append(process.pid)

progress\_made = True

if not progress\_made:

break

return finish, safe\_sequence

def detect\_deadlock(self):

finish, safe\_sequence = self.is\_safe()

if all(finish):

return False, safe\_sequence # No deadlock

else:

return True, [] # Deadlock detected

class DeadlockApp:

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

self.root = root

self.root.title("Deadlock Detection and Recovery System")

self.processes = []

self.available\_resources = []

self.create\_widgets()

def create\_widgets(self):

tk.Label(self.root, text="Number of Resources:").grid(row=0, column=0)

self.resource\_entry = tk.Entry(self.root)

self.resource\_entry.grid(row=0, column=1)

tk.Label(self.root, text="Available Resources (comma-separated):").grid(row=1, column=0)

self.available\_entry = tk.Entry(self.root)

self.available\_entry.grid(row=1, column=1)

tk.Button(self.root, text="Set Resources", command=self.set\_resources).grid(row=2, columnspan=2)

tk.Label(self.root, text="Process ID:").grid(row=3, column=0)

self.pid\_entry = tk.Entry(self.root)

self.pid\_entry.grid(row=3, column=1)

tk.Label(self.root, text="Max Resources (comma-separated):").grid(row=4, column=0)

self.max\_entry = tk.Entry(self.root)

self.max\_entry.grid(row=4, column=1)

tk.Button(self.root, text="Add Process", command=self.add\_process).grid(row=5, columnspan=2)

tk.Button(self.root, text="Check Deadlock", command=self.check\_deadlock).grid(row=6, columnspan=2)

def set\_resources(self):

resources = self.available\_entry.get().split(',')

self.available\_resources = [int(r) for r in resources]

def add\_process(self):

pid = self.pid\_entry.get()

max\_resources = self.max\_entry.get().split(',')

max\_resources = [int(r) for r in max\_resources]

self.processes.append(Process(pid, max\_resources))

def check\_deadlock(self):

detector = DeadlockDetector(self.processes, self.available\_resources)

deadlock, safe\_sequence = detector.detect\_deadlock()

if deadlock:

deadlock\_processes = [process for process in self.processes if not all(process.need[j] <= self.available\_resources[j] for j in range(len(self.available\_resources)))]

self.show\_recovery\_options(deadlock\_processes)

else:

messagebox.showinfo("No Deadlock", f"System is in a safe state. Safe sequence: {safe\_sequence}")

def show\_recovery\_options(self, deadlock\_processes):

# Show the reason for deadlock

messagebox.showinfo("Deadlock Detected", "Deadlock has been detected!")

# Create a new window for recovery options

recovery\_window = tk.Toplevel(self.root)

recovery\_window.title("Deadlock Recovery Options")

tk.Label(recovery\_window, text="Select a process to terminate:").pack()

for process in deadlock\_processes:

tk.Button(recovery\_window, text=process.pid, command=lambda p=process: self.terminate\_process(p)).pack()

tk.Label(recovery\_window, text="Select a process to preempt resources from:").pack()

for process in deadlock\_processes:

tk.Button(recovery\_window, text=f"Preempt from {process.pid}", command=lambda p=process: self.preempt\_resources(p)).pack()

tk.Button(recovery\_window, text="Cancel", command=recovery\_window.destroy).pack()

def terminate\_process(self, process):

# Remove the process from the system

self.processes.remove(process)

messagebox.showinfo("Process Terminated", f"Process {process.pid} has been terminated.")

def preempt\_resources(self, process):

# Show current allocation and allow user to specify how many resources to preempt

preemption\_window = tk.Toplevel(self.root)

preemption\_window.title(f"Preempt Resources from {process.pid}")

tk.Label(preemption\_window, text="Current Allocation:").pack()

tk.Label(preemption\_window, text=f"Resource Allocation: {process.allocated}").pack()

tk.Label(preemption\_window, text="Enter number of resources to preempt (comma-separated):").pack()

preempt\_entry = tk.Entry(preemption\_window)

preempt\_entry.pack()

def confirm\_preemption():

preempt\_values = list(map(int, preempt\_entry.get().split(',')))

for i in range(len(process.allocated)):

if preempt\_values[i] > process.allocated[i]:

messagebox.showerror("Error", "Cannot preempt more resources than allocated.")

return

# Update the allocated and available resources

process.allocated[i] -= preempt\_values[i]

self.available\_resources[i] += preempt\_values[i]

messagebox.showinfo("Resources Preempted", f"Resources preempted from {process.pid}.")

preemption\_window.destroy()

tk.Button(preemption\_window, text="Confirm Preemption", command=confirm\_preemption).pack()

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

root = tk.Tk()

app = DeadlockApp(root)

root.mainloop()

***THANKS***