Nagar Yuwak Shikshan Sanstha's

Yeshwantrao Chavan College of Engineering

(An Autonomous Institution affiliated to Rashtrasant Tukadoji Maharaj Nagpur University)
Hingna Road, Wanadongri, Nagpur - 441 110
NAAC A++
Ph.: 07104-237919, 234623, 329249, 329250 Fax: 07104-232376, Website: www.ycce.edu

Department of Computer Technology

Vision of the Department

To be a well-known centre for pursuing computer education through innovative pedagogy, value-based education and industry collaboration.

Mission of the Department

To establish learning ambience for ushering in computer engineering professionals in core and multidisciplinary area by developing Problem-solving skills through emerging technologies.

Session 2025-2026

Vision: Dream of where you want.	**Mission:** Means to achieve Vision

Program Educational Objectives of the program (PEO): (broad statements that describe the professional and career accomplishments)

PEO1	**Preparation**	**P: Preparation**	**Pep-CL abbreviation pronounce as Pep-si-lL easy to recall**
PEO2	**Core Competence**	**E: Environment (Learning Environment)**	
PEO3	**Breadth**	**P: Professionalism**	
PEO4	**Professionalism**	**C: Core Competence**	
PEO5	**Learning Environment**	**L: Breadth (Learning in diverse areas)**	

Program Outcomes (PO): (statements that describe what a student should be able to do and know by the end of a program)

Keywords of POs:
Engineering knowledge, Problem analysis, Design/development of solutions, Conduct Investigations of Complex Problems, Engineering Tool Usage, The Engineer and The World, Ethics, Individual and Collaborative Team work, Communication, Project Management and Finance, Life-Long Learning

PSO Keywords: Cutting edge technologies, Research

"I am an engineer, and I know how to apply engineering knowledge to investigate, analyse and design solutions to complex problems using tools for entire world following all ethics in a collaborative way with proper management skills throughout my life." *to contribute to the development of cutting-edge technologies and Research*.

Integrity: I will adhere to the Laboratory Code of Conduct and ethics in its entirety.

Name and Signature of Student and Date
Bhushan Tayade
30/10/2025

Session	**2025-26 (ODD)**	**Course Name**	**Operating System Lab**
Semester	**5**	**Course Code**	**23IOT1504**
Roll No	**35**	**Name of Student**	**Bhushan Tayade**

Practical Number	**5**
Course Outcome	1. Understand Computer System Configuration and Simulate system resources efficiently using Linux Commands (CO1) 2. Analyse operating system functionalities utilizing system calls, thread programming and process scheduling algorithms (CO2) 3. Apply Synchronization primitives to implement a Deadlock-free solution(CO3) 4. Simulate Disk scheduling, Memory allocation, File allocation, page replacement algorithms (CO4)
Aim	**Stimulate bankers deadlock detection algorithm.**
Problem Definition	
Theory (100 words)	The code simulates the Banker's Algorithm, a classic deadlock avoidance technique in Operating Systems. The algorithm works by examining the state of resource allocation to ensure no sequence of resource requests can lead to a deadlock. It utilizes three key matrices: Allocation (resources currently held), Max (maximum resources needed), and Need (Max - Allocation, remaining resources needed). It also uses an Available vector. The Safety Algorithm checks if the system is in a safe state by finding a sequence of processes that can finish. If no safe sequence exists, the system is unsafe. The Deadlock Detection Algorithm (used when an unsafe state is reached) runs a similar check to identify the specific processes that are permanently blocked (deadlocked) because their Need cannot be met by the current Available resources plus the allocations of finished processes.

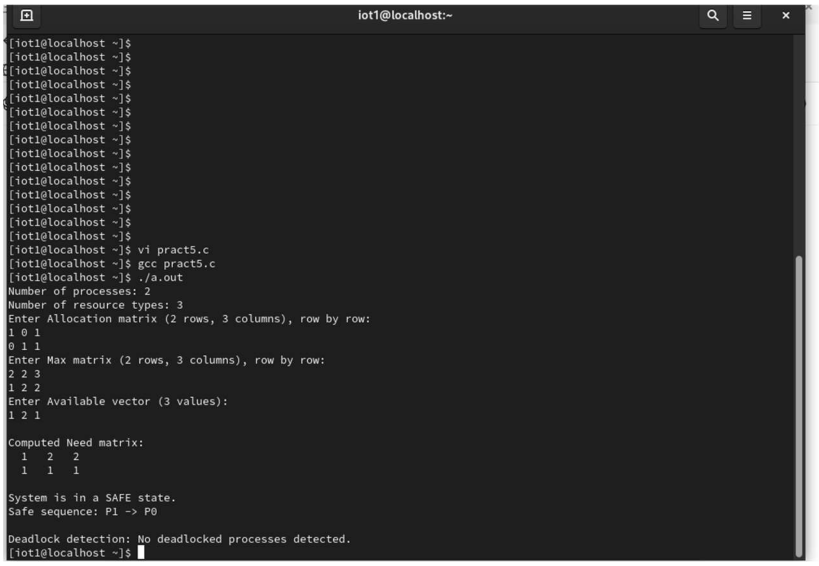
Procedure and Execution (100 Words)	Step for Implementation: 1. Start the program and input the total number of processes (n) and resource types (m). ○ These values define the size of all matrices (Allocation, Max, Need, and Available). 2. Enter the Allocation matrix, which shows how many instances of each resource are currently allocated to each process. 3. Enter the Max matrix, which shows the maximum demand of each process for each resource. 4. Enter the Available vector, which shows how many instances of each resource are currently available in the system. 5. Calculate the Need matrix using the formula: $\text{Need}[i][j] = \text{Max}[i][j] - \text{Allocation}[i][j]$ This represents how many more resources each process still requires to complete its execution. 6. Apply the Banker's Safety Algorithm to check if the system is in a safe state: ○ Initialize Work = Available and Finish[i] = false for all processes. ○ Find a process whose Need ≤ Work. ○ If found, assume it finishes and release its allocated resources (Work = Work + Allocation[i]). ○ Repeat until all processes can finish or no further progress is possible. ○ If all processes finish, the system is safe and the program prints the safe sequence. 7. If the system is unsafe, the program then performs Deadlock Detection: ○ Initialize Work = Available and set Finish[i] = true for processes with zero allocation. ○ For remaining processes, check if their current requests (Need) can be satisfied by Work. ○ If yes, simulate their completion by adding their Allocation to Work. ○ Repeat until no more progress can be made. ○ Processes that still have Finish[i] = false at the end are deadlocked. 8. Display the results: ○ Print whether the system is safe or unsafe. ○ If unsafe, display the list of deadlocked processes. 9. End the program.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

void read_matrix(int n, int m, int mat[n][m], const char *name) {
   printf("Enter %s matrix (%d rows, %d columns), row by row:\n", name, n, m);
   for (int i = 0; i < n; ++i) {
      for (int j = 0; j < m; ++j) {
         if (scanf("%d", &mat[i][j]) != 1) {
            fprintf(stderr, "Input error reading %s[%d][%d]\n", name, i, j);
            exit(EXIT_FAILURE);
         }
         if (mat[i][j] < 0) {
            fprintf(stderr, "Negative values not allowed; found %d\n", mat[i][j]);
            exit(EXIT_FAILURE);
         }
      }
   }
}


void read_vector(int m, int vec[m], const char *name) {
   printf("Enter %s vector (%d values):\n", name, m);
   for (int j = 0; j < m; ++j) {
      if (scanf("%d", &vec[j]) != 1) {
         fprintf(stderr, "Input error reading %s[%d]\n", name, j);
         exit(EXIT_FAILURE);
      }
      if (vec[j] < 0) {
         fprintf(stderr, "Negative values not allowed; found %d\n", vec[j]);
         exit(EXIT_FAILURE);
      }
```

```
    }
}

void compute_need(int n, int m, int need[n][m], int max[n][m], int
alloc[n][m]) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            need[i][j] = max[i][j] - alloc[i][j];
            if (need[i][j] < 0) {
                fprintf(stderr, "Error: Allocation > Max for process %d resource
%d\n", i, j);
                exit(EXIT_FAILURE);
            }
        }
    }
}

bool bankers_safety(int n, int m, int alloc[n][m], int need[n][m], int
available[m], int safe_seq_out[n]) {
    int work[m];
    bool finish[n];

    for (int j = 0; j < m; ++j) work[j] = available[j];
    for (int i = 0; i < n; ++i) finish[i] = false;

    int count = 0;
    while (count < n) {
        bool found = false;
        for (int i = 0; i < n; ++i) {
            if (!finish[i]) {
                bool can_allocate = true;
                // Check if Need[i] <= Work
                for (int j = 0; j < m; ++j) {
                    if (need[i][j] > work[j]) {
                        can_allocate = false;
                        break;
                    }
                }

                if (can_allocate) {
                    // Work = Work + Allocation[i]
                    for (int j = 0; j < m; ++j) work[j] += alloc[i][j];
                    safe_seq_out[count++] = i;
                    finish[i] = true;
                    found = true;
                }
            }
        }
        if (!found) break; // no further process can proceed
    }

    if (count == n) return true;
    return false;
}

int deadlock_detection(int n, int m, int alloc[n][m], int request[n][m], int
available[m], int deadlocked_out[n]) {
    int work[m];
    bool finish[n];
    bool progress = true;

    for (int j = 0; j < m; ++j) work[j] = available[j];
    for (int i = 0; i < n; ++i) {
        bool all_zero = true;
        for (int j = 0; j < m; ++j) {
            if (alloc[i][j] != 0) {
                all_zero = false;
                break;
            }
        }
        finish[i] = all_zero ? true : false;
    }

    do {
        progress = false;
        for (int i = 0; i < n; ++i) {
            if (!finish[i]) {
                bool can_satisfy = true;
                // Check if Request[i] <= Work
                for (int j = 0; j < m; ++j) {
                    if (request[i][j] > work[j]) {
                        can_satisfy = false;
                        break;
                    }
                }

                if (can_satisfy) {
                    // 3. Process can finish; release its allocation
                    // Work = Work + Allocation[i]
                    for (int j = 0; j < m; ++j) work[j] += alloc[i][j];
                    finish[i] = true;
                    progress = true;
                }
            }
        }
    } while (progress);

    // 4. Check for deadlocked processes
    int dead_count = 0;
    for (int i = 0; i < n; ++i) {
        if (!finish[i]) {
            deadlocked_out[dead_count++] = i;
        }
    }
    return dead_count;
}

int main() {
    int n, m; // n: number of processes, m: number of resource types

    printf("Number of processes: ");
    if (scanf("%d", &n) != 1 || n <= 0) {
        fprintf(stderr, "Invalid number of processes\n");
        return 1;
    }
    printf("Number of resource types: ");
    if (scanf("%d", &m) != 1 || m <= 0) {
        fprintf(stderr, "Invalid number of resource types\n");
        return 1;
    }

    int alloc[n][m];
    int max[n][m];
    int need[n][m];
    int available[m];
    int safe_seq[n];
    int deadlocked[n];

    read_matrix(n, m, alloc, "Allocation");
    read_matrix(n, m, max, "Max");
    read_vector(m, available, "Available");
    compute_need(n, m, need, max, alloc);

    printf("\nComputed Need matrix:\n");
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) printf("%3d ", need[i][j]);
        printf("\n");
    }

    bool safe = bankers_safety(n, m, alloc, need, available, safe_seq);

    if (safe) {
        printf("\nSystem is in a SAFE state. \\nSafe sequence: ");
        for (int i = 0; i < n; ++i) {
            printf("P%d", safe_seq[i]);
            if (i < n - 1) printf(" -> ");
        }
        printf("\n");
    } else {
        printf("\nSystem is NOT in a safe state (UNSAFE).\n");
    }

    int dead_count = deadlock_detection(n, m, alloc, need, available,
deadlocked);

    printf("\nDeadlock Detection: ");
    if (dead_count == 0) {
        printf("No deadlocked processes detected.\n");
    } else {
        printf("%d deadlocked process(es): ", dead_count);
        for (int k = 0; k < dead_count; ++k) {
            printf("P%d", deadlocked[k]);
            if (k < dead_count - 1) printf(", ");
        }
        printf("\n");
    }
    printf("\n");

    return 0;
}
```

Output:

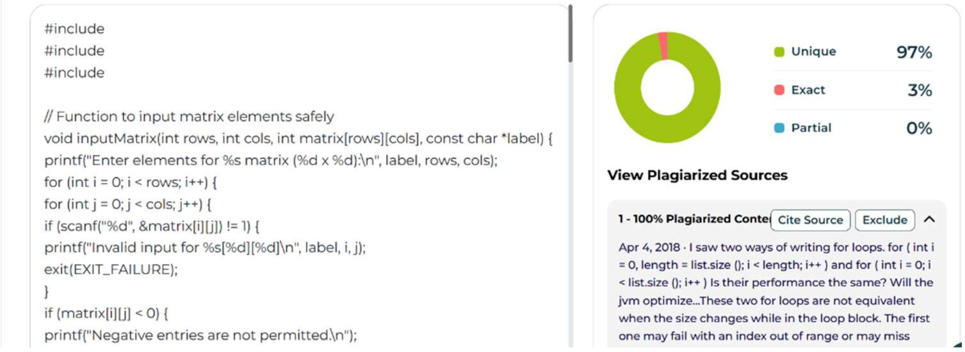


```
[iot1@localhost ~]$
[iot1@localhost ~]$
[iot1@localhost ~]$
[iot1@localhost ~]$
[iot1@localhost ~]$
[iot1@localhost ~]$
[iot1@localhost ~]$
[iot1@localhost ~]$
[iot1@localhost ~]$
[iot1@localhost ~]$
[iot1@localhost ~]$
[iot1@localhost ~]$
[iot1@localhost ~]$
[iot1@localhost ~]$
[iot1@localhost ~]$
[iot1@localhost ~]$ vi pract5.c
[iot1@localhost ~]$ gcc pract5.c
[iot1@localhost ~]$ ./a.out
Number of processes: 2
Number of resource types: 3
Enter Allocation matrix (2 rows, 3 columns), row by row:
1 0 1
0 1 1
Enter Max matrix (2 rows, 3 columns), row by row:
2 2 3
1 2 2
Enter Available vector (3 values):
1 2 1

Computed Need matrix:
  1   2   2
  1   1   1

System is in a SAFE state.
Safe sequence: P1 -> P0

Deadlock detection: No deadlocked processes detected.
[iot1@localhost ~]$
```

Output Analysis

- The program first takes input for number of processes, resources, Allocation, Max, and Available matrices.
- It then calculates the Need matrix using Need = Max - Allocation.
- The Banker's Safety Algorithm checks if the system is in a safe state.
- If safe, it prints the safe sequence showing the order of process execution.
- If not safe, it declares the system UNSAFE.

	• The Deadlock Detection Algorithm is then applied to find any deadlocked processes. • Finally, the program displays either "No deadlock detected" or lists the deadlocked processes.
Link of student Github profile where lab assignment has been uploaded	"https://github.com/Bhushan-Tayade/YCCN-23071391.git"
Conclusion	The program is executed successfully.

Plag Report (Similarity index < 12%)	
	#include
	#include
	#include
	
	// Function to input matrix elements safely
	void inputMatrix(int rows, int cols, int matrix[rows][cols], const char *label) {
	printf("Enter elements for %s matrix (%d x %d):\n", label, rows, cols);
	for (int i = 0; i < rows; i++) {
	for (int j = 0; j < cols; j++) {
	if (scanf("%d", &matrix[i][j]) != 1) {
	printf("Invalid input for %s[%d][%d]\n", label, i, j);
	exit(EXIT_FAILURE);
	}
	if (matrix[i][j] < 0) {
	printf("Negative entries are not permitted.\n");
	
	🟩 Unique 97%
	🟥 Exact 3%
	🟦 Partial 0%
	View Plagiarized Sources
	1 - 100% Plagiarized Conte Cite Source Exclude ^
	Apr 4, 2018 · I saw two ways of writing for loops. for (int i = 0, length = list.size (); i < length; i++) and for (int i = 0; i < list.size (); i++) Is their performance the same? Will the jvm optimize...These two for loops are not equivalent when the size changes while in the loop block. The first one may fail with an index out of range or may miss

Date	30/10/2025