Ex.No: 3B	Implementation of Distributed Deadlock Detection Algorithm
Date: 05.02.2024	

Code:

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

#define MAX_PROCESSES 10
#define MAX_RESOURCES 10

int available[MAX_RESOURCES];
int max[MAX_PROCESSES][MAX_RESOURCES];
int allocation[MAX_PROCESSES][MAX_RESOURCES];
int need[MAX_PROCESSES][MAX_RESOURCES];
bool finish[MAX_PROCESSES];

int num_processes, num_resources;

// Function to check if the system is in a safe state
bool isSafe() {
    int work[MAX_RESOURCES];
    bool finish_copy[MAX_PROCESSES];

    for (int i = 0; i < num_resources; i++)
        work[i] = available[i];

    for (int i = 0; i < num_processes; i++)
        finish_copy[i] = finish[i];

    int count = 0;
    while (count < num_processes) {
        bool found = false;
        for (int i = 0; i < num_processes; i++) {
            if (!finish_copy[i]) {
                int j;
                for (j = 0; j < num_resources; j++) {
                    if (need[i][j] > work[j])
                        break;
                }
                if (j == num_resources) {
                    for (int k = 0; k < num_resources; k++)
                        work[k] += allocation[i][k];
                    finish_copy[i] = true;
                    found = true;
                    count++;
                }
```

```
            }
        }
        if (!found)
            return false; // If no process can be executed, the system is not in a safe state
    }
    return true; // If all processes can be executed, the system is in a safe state
}

// Function to check for deadlock
bool detectDeadlock() {
    for (int i = 0; i < num_processes; i++) {
        if (!finish[i]) {
            bool canExecute = true;
            for (int j = 0; j < num_resources; j++) {
                if (need[i][j] > available[j]) {
                    canExecute = false;
                    break;
                }
            }
            if (canExecute)
                return false; // If there is a process that can execute, there's no deadlock
        }
    }
    return true; // If no process can execute, there's a deadlock
}

// Function to simulate resource request by a process
void requestResources(int process_id, int request[]) {
    for (int i = 0; i < num_resources; i++) {
        if (request[i] > need[process_id][i] || request[i] > available[i]) {
            printf("Error: Invalid request by process %d\n", process_id);
            return;
        }
    }

    for (int i = 0; i < num_resources; i++) {
        available[i] -= request[i];
        allocation[process_id][i] += request[i];
        need[process_id][i] -= request[i];
    }

    if (!isSafe()) {
        // If granting the request results in an unsafe state, rollback
        for (int i = 0; i < num_resources; i++) {
            available[i] += request[i];
            allocation[process_id][i] -= request[i];
            need[process_id][i] += request[i];
        }
```

```
        printf("Deadlock detected after granting request by process %d\n", process_id);
    } else {
        printf("Request by process %d granted successfully\n", process_id);
    }
}

// Function to simulate resource release by a process
void releaseResources(int process_id, int release[]) {
    for (int i = 0; i < num_resources; i++) {
        if (release[i] > allocation[process_id][i]) {
            printf("Error: Invalid release by process %d\n", process_id);
            return;
        }
    }

    for (int i = 0; i < num_resources; i++) {
        available[i] += release[i];
        allocation[process_id][i] -= release[i];
        need[process_id][i] += release[i];
    }

    printf("Resources released by process %d\n", process_id);
}

int main() {
    // Input number of processes and resources
    printf("Enter the number of processes: ");
    scanf("%d", &num_processes);
    printf("Enter the number of resources: ");
    scanf("%d", &num_resources);

    // Input available resources
    printf("Enter the available resources: ");
    for (int i = 0; i < num_resources; i++)
        scanf("%d", &available[i]);

    // Input maximum resources for each process
    printf("Enter the maximum resources for each process:\n");
    for (int i = 0; i < num_processes; i++) {
        printf("Process %d: ", i);
        for (int j = 0; j < num_resources; j++)
            scanf("%d", &max[i][j]);
    }

    // Input allocated resources for each process
    printf("Enter the allocated resources for each process:\n");
    for (int i = 0; i < num_processes; i++) {
        printf("Process %d: ", i);
```

```
        for (int j = 0; j < num_resources; j++) {
            scanf("%d", &allocation[i][j]);
            need[i][j] = max[i][j] - allocation[i][j];
        }
    }

    // Initialize finish array
    for (int i = 0; i < num_processes; i++)
        finish[i] = false;

    // Detect and handle deadlock
    if (detectDeadlock()) {
        printf("Deadlock detected\n");
        // Handle deadlock here
        // For example, kill processes or roll back transactions
    } else {
        printf("No deadlock detected\n");
    }

    return 0;
}
```

Output:

```
swetha@Swethas-MacBook-Air Desktop % ./a.out
Enter the number of processes: 5
Enter the number of resources: 3
Enter the available resources: 3 3 2
Enter the maximum resources for each process:
Process 0: 7 5 3
Process 1: 3 2 2
Process 2: 9 0 2
Process 3: 2 2 2
Process 4: 4 3 3
Enter the allocated resources for each process:
Process 0: 0 1 0
Process 1: 2 0 0
Process 2: 3 0 2
Process 3: 2 1 1
Process 4: 0 0 2
No deadlock detected
swetha@Swethas-MacBook-Air Desktop %
```

Figure 1: Deadlock detection using distributed deadlock detection algorithm