Bansilal Ramnath Agarwal Charitable Trust's

Vishwakarma Institute of Technology

(An Autonomous Institute affiliated to Savitribai Phule Pune University)

Assignment-6

Subject	Operating System
Name	Akash Bhandari
Class	CS-A
Roll No.	06

Implement following algorithms

1. Deadlock Avoidance

2. Deadlock Detection

1. Deadlock Avoidance

Code :-

```
#include <stdio.h>
#define MAX_PROCESSES 100
#define MAX_RESOURCES 100

int available[MAX_RESOURCES];
int maximum[MAX_PROCESSES][MAX_RESOURCES];
int allocation[MAX_PROCESSES][MAX_RESOURCES];
int need[MAX_PROCESSES][MAX_RESOURCES];
int num_processes;
```

```
int num_resources;

void calculate_need() {
    int i, j;
    for (i = 0; i < num_processes; i++) {
        for (j = 0; j < num_resources; j++) {
            need[i][j] = maximum[i][j] - allocation[i][j];
        }
    }
}

int is_safe() {
    int i, j;
    int work[MAX_RESOURCES];
    int finish[MAX_PROCESSES];

    // Initialize work and finish arrays
    for (i = 0; i < num_resources; i++) {
        work[i] = available[i];
    }
    for (i = 0; i < num_processes; i++) {
        finish[i] = 0;
    }

    // Find a process that can finish
    int found = 1;
    while (found) {
        found = 0;
        for (i = 0; i < num_processes; i++) {
            if (!finish[i]) {
                int can_finish = 1;
                for (j = 0; j < num_resources; j++) {
                    if (need[i][j] > work[j]) {
                        can_finish = 0;
                        break;
                    }
                }
                if (can_finish) {
                    found = 1;
                    finish[i] = 1;
                    for (j = 0; j < num_resources; j++) {
                        work[j] += allocation[i][j];
                    }
                }
            }
        }
    }
```

```
    // Check if all processes finished
    for (i = 0; i < num_processes; i++) {
        if (!finish[i]) {
            return 0;
        }
    }
    return 1;
}

void detect_deadlock() {
    int i, j;

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

    // Input available array
    printf("Enter the available array:\n");
    for (i = 0; i < num_resources; i++) {
        scanf("%d", &available[i]);
    }

    // Input maximum and allocation arrays
    printf("Enter the maximum matrix:\n");
    for (i = 0; i < num_processes; i++) {
        for (j = 0; j < num_resources; j++) {
            scanf("%d", &maximum[i][j]);
        }
    }
    printf("Enter the allocation matrix:\n");
    for (i = 0; i < num_processes; i++) {
        for (j = 0; j < num_resources; j++) {
            scanf("%d", &allocation[i][j]);
        }
    }

    calculate_need();
    if (is_safe()) {
        printf("No deadlock detected.\n");
    } else {
        printf("Deadlock detected.\n");
    }
}

int main() {
    detect_deadlock();
```

```
    return 0;
}
```

Output :-

```
Enter the number of processes: 5
Enter the number of resources: 3
Enter the available array:
5       3       3
Enter the allocation matrix:
0       1       0
2       0       0
3       0       2
2       1       1
0       0       2
No deadlock detected.
```

2. Deadlock detection

Code :-

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_PROCESSES 100
#define MAX_RESOURCES 100

int available[MAX_RESOURCES];
int allocation[MAX_PROCESSES][MAX_RESOURCES];
int request[MAX_PROCESSES][MAX_RESOURCES];
int num_processes;
int num_resources;

int is_safe() {
    int i, j;
    int avai_resources[MAX_RESOURCES];
    int state[MAX_PROCESSES];
    for (i = 0; i < num_resources; i++) {
        avai_resources[i] = available[i];
    }
```

```
    for (i = 0; i < num_processes; i++) {
        state[i] = 0;
    }
    int found = 1;
    while (found) {
        found = 0;
        for (i = 0; i < num_processes; i++) {
            if (!state[i]) {
                int can_finish = 1;
                for (j = 0; j < num_resources; j++) {
                    if (request[i][j] > avai_resources[j]) {
                        can_finish = 0;
                        break;
                    }
                }
                if (can_finish) {
                    found = 1;
                    state[i] = 1;
                    for (j = 0; j < num_resources; j++) {
                        avai_resources[j] += allocation[i][j];
                    }
                }
            }
        }
    }

    for (i = 0; i < num_processes; i++) {
        if (!state[i]) {
            return 0;
        }
    }
    return 1;
}

void detect_deadlock() {
    if (is_safe()) {
        printf("No deadlock detected.\n");
    } else {
        printf("Deadlock detected.\n");
    }
}

int main() {
    int i, j;
    printf("Enter the number of processes: ");
    scanf("%d", &num_processes);
    printf("Enter the number of resources: ");
    scanf("%d", &num_resources);
```

```
    printf("Enter the available resources: ");
    for (i = 0; i < num_resources; i++) {
        scanf("%d", &available[i]);
    }
    printf("Enter the allocation matrix:\n");
    for (i = 0; i < num_processes; i++) {
        printf("Process %d: ", i);
        for (j = 0; j < num_resources; j++) {
            scanf("%d", &allocation[i][j]);
        }
    }
    printf("Enter the request matrix:\n");
    for (i = 0; i < num_processes; i++) {
        printf("Process %d: ", i);
        for (j = 0; j < num_resources; j++) {
            scanf("%d", &request[i][j]);
        }
    }
    detect_deadlock();

    return 0;
}
```

Output :-

```
Enter the number of processes: 5
Enter the number of resources: 3
Enter the available array:
5       3       3
Enter the available resources: 0        0       0
Enter the allocation matrix:
Process 0: 0    1       0
Process 1: 2    0       0
Process 2: 3    0       3
Process 3: 2    1       1
Process 4: 0    0       2
Enter the request matrix:
Process 0: 0    0       0
Process 1: 2    0       2
Process 2: 0    0       0
Process 3: 0    0       0
Process 4: 0    0       2
No deadlock detected.
```