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** | Ajinkya Walunj |
| **Class** | CS-A |
| **Roll No.** | 05 |

**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");

    }

}

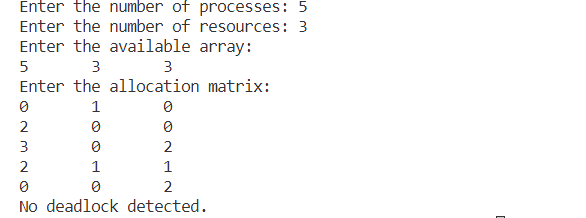
int main() {

    detect\_deadlock();

    return 0;

}

*Output :-*



1. **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 :-*

