**Assignment No: 3 Deadlock detection and avoidance**

**SET A**

**1. Write the program to calculate minimum number of resources needed to avoid**

**Deadlock.**

**Ans**

#include <stdio.h>

#define MAX\_PROCESSES 10

#define MAX\_RESOURCES 10

int max[MAX\_PROCESSES][MAX\_RESOURCES];

int allot[MAX\_PROCESSES][MAX\_RESOURCES];

int need[MAX\_PROCESSES][MAX\_RESOURCES];

int available[MAX\_RESOURCES];

void calculateNeed(int processes[], int m, int n) {

for (int i = 0; i < m; i++)

for (int j = 0; j < n; j++)

need[i][j] = max[i][j] - allot[i][j];

}

int isSafe(int processes[], int m, int n) {

int finish[MAX\_PROCESSES] = {0};

int safeSeq[MAX\_PROCESSES];

int work[MAX\_RESOURCES];

for (int i = 0; i < n; i++)

work[i] = available[i];

int count = 0;

while (count < m) {

int found = 0;

for (int p = 0; p < m; p++) {

if (!finish[p]) {

int j;

for (j = 0; j < n; j++)

if (need[p][j] > work[j])

break;

if (j == n) {

for (int k = 0; k < n; k++)

work[k] += allot[p][k];

safeSeq[count++] = p;

finish[p] = 1;

found = 1;

}

}

}

if (!found) {

printf("No safe sequence exists.\n");

return 0;

}

}

return 1;

}

int main() {

int processes[] = { 0, 1, 2, 3, 4 };

int m = 5;

int n;

printf("Enter the number of resources: ");

scanf("%d", &n);

printf("Enter the maximum resources matrix:\n");

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) {

scanf("%d", &max[i][j]);

}

}

printf("Enter the allocated resources matrix:\n");

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) {

scanf("%d", &allot[i][j]);

}

}

printf("Enter the available resources:\n");

for (int i = 0; i < n; i++) {

scanf("%d", &available[i]);

}

calculateNeed(processes, m, n);

if (isSafe(processes, m, n)) {

printf("System is in a safe state.\n");

} else {

printf("System is not in a safe state.\n");

}

int min\_resources\_needed = 0;

for (int j = 0; j < n; j++) {

int max\_need = 0;

for (int i = 0; i < m; i++) {

max\_need += need[i][j];

}

min\_resources\_needed += max\_need;

}

printf("Minimum additional resources needed to avoid deadlock: %d\n", min\_resources\_needed);

return 0;

}

**Output:**

Enter the number of resources: 3

Enter the maximum resources matrix:

7 5 3

3 2 2

9 0 2

2 2 2

4 3 3

Enter the allocated resources matrix:

0 1 0

2 0 0

3 0 2

2 1 1

0 0 2

Enter the available resources:

3 3 2

System is in a safe state.

Minimum additional resources needed to avoid deadlock: 35

**2. Write a C program to accept the number of process and resources and find the need**

**matrix content and display it.**

**Ans**

#include <stdio.h>

#define MAX\_PROCESSES 10

#define MAX\_RESOURCES 10

int max[MAX\_PROCESSES][MAX\_RESOURCES];

int allot[MAX\_PROCESSES][MAX\_RESOURCES];

int need[MAX\_PROCESSES][MAX\_RESOURCES];

void calculateNeed(int m, int n) {

for (int i = 0; i < m; i++)

for (int j = 0; j < n; j++)

need[i][j] = max[i][j] - allot[i][j];

}

void displayNeedMatrix(int m, int n) {

printf("Need Matrix:\n");

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) {

printf("%d ", need[i][j]);

}

printf("\n");

}

}

int main() {

int m, n;

printf("Enter the number of processes: ");

scanf("%d", &m);

printf("Enter the number of resources: ");

scanf("%d", &n);

printf("Enter the maximum resources matrix:\n");

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) {

printf("Max[%d][%d]: ", i, j);

scanf("%d", &max[i][j]);

}

}

printf("Enter the allocated resources matrix:\n");

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) {

printf("Allot[%d][%d]: ", i, j);

scanf("%d", &allot[i][j]);

}

}

calculateNeed(m, n);

displayNeedMatrix(m, n);

return 0;

}

**Output**

Enter the number of processes: 3

Enter the number of resources: 2

Enter the maximum resources matrix:

Max[0][0]: 2

Max[0][1]: 1

Max[1][0]: 3

Max[1][1]: 4

Max[2][0]: 4

Max[2][1]: 3

Enter the allocated resources matrix:

Allot[0][0]: 1

Allot[0][1]: 3

Allot[1][0]: 2

Allot[1][1]: 4

Allot[2][0]: 2

Allot[2][1]: 1

Need Matrix:

1 -2

1 0

2 2

**SET B**

**1. Partially implement the Menu driven Banker’s algorithm for accepting Allocation,**

**Max from user.**

**a) Accept Available**

**b) Display Allocation, Max**

**c) Find Need and Display It,**

**d) Display Available**

**Ans: $ touch bankeralgoB1.c**

#include <stdio.h>

#define MAX\_PROCESSES 10

#define MAX\_RESOURCES 10

int max[MAX\_PROCESSES][MAX\_RESOURCES];

int allot[MAX\_PROCESSES][MAX\_RESOURCES];

int need[MAX\_PROCESSES][MAX\_RESOURCES];

int available[MAX\_RESOURCES];

int m, n;

void calculateNeed() {

for (int i = 0; i < m; i++)

for (int j = 0; j < n; j++)

need[i][j] = max[i][j] - allot[i][j];

}

void displayAllocation() {

printf("Allocation Matrix:\n");

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) {

printf("%d ", allot[i][j]);

}

printf("\n");

}

}

void displayMax() {

printf("Max Matrix:\n");

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) {

printf("%d ", max[i][j]);

}

printf("\n");

}

}

void displayNeed() {

calculateNeed();

printf("Need Matrix:\n");

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) {

printf("%d ", need[i][j]);

}

printf("\n");

}

}

void displayAvailable() {

printf("Available Resources:\n");

for (int i = 0; i < n; i++) {

printf("%d ", available[i]);

}

printf("\n");

}

int main() {

int choice;

printf("Enter the number of processes: ");

scanf("%d", &m);

printf("Enter the number of resources: ");

scanf("%d", &n);

while (1) {

printf("\nMenu:\n");

printf("1. Accept Allocation Matrix\n");

printf("2. Accept Max Matrix\n");

printf("3. Accept Available Resources\n");

printf("4. Display Allocation Matrix\n");

printf("5. Display Max Matrix\n");

printf("6. Display Need Matrix\n");

printf("7. Display Available Resources\n");

printf("8. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the Allocation Matrix:\n");

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) {

printf("Allot[%d][%d]: ", i, j);

scanf("%d", &allot[i][j]);

}

}

break;

case 2:

printf("Enter the Max Matrix:\n");

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) {

printf("Max[%d][%d]: ", i, j);

scanf("%d", &max[i][j]);

}

}

break;

case 3:

printf("Enter the Available Resources:\n");

for (int i = 0; i < n; i++) {

printf("Available[%d]: ", i);

scanf("%d", &available[i]);

}

break;

case 4:

displayAllocation();

break;

case 5:

displayMax();

break;

case 6:

displayNeed();

break;

case 7:

displayAvailable();

break;

case 8:

return 0;

default:

printf("Invalid choice. Please try again.\n");

}

}

return 0;

}

**Output :**

**$ gcc -o bank bankeralgoB1.c**

**$ ./bank**

Enter the number of processes: 3

Enter the number of resources: 2

Menu:

1. Accept Allocation Matrix

2. Accept Max Matrix

3. Accept Available Resources

4. Display Allocation Matrix

5. Display Max Matrix

6. Display Need Matrix

7. Display Available Resources

8. Exit

Enter your choice: 1

Enter the Allocation Matrix:

Allot[0][0]: 2

Allot[0][1]: 1

Allot[1][0]: 2

Allot[1][1]: 3

Allot[2][0]: 4

Allot[2][1]: 5

Menu:

1. Accept Allocation Matrix

2. Accept Max Matrix

3. Accept Available Resources

4. Display Allocation Matrix

5. Display Max Matrix

6. Display Need Matrix

7. Display Available Resources

8. Exit

Enter your choice: 1

Enter the Allocation Matrix:

Allot[0][0]: 3

Allot[0][1]: 1

Allot[1][0]: 2

Allot[1][1]: 1

Allot[2][0]: 2

Allot[2][1]: 3

Menu:

1. Accept Allocation Matrix

2. Accept Max Matrix

3. Accept Available Resources

4. Display Allocation Matrix

5. Display Max Matrix

6. Display Need Matrix

7. Display Available Resources

8. Exit

Enter your choice: 3

Enter the Available Resources:

Available[0]: 1

Available[1]: 2

Menu:

1. Accept Allocation Matrix

2. Accept Max Matrix

3. Accept Available Resources

4. Display Allocation Matrix

5. Display Max Matrix

6. Display Need Matrix

7. Display Available Resources

8. Exit

Enter your choice: 4

Allocation Matrix:

3 1

2 1

2 3

Menu:

1. Accept Allocation Matrix

2. Accept Max Matrix

3. Accept Available Resources

4. Display Allocation Matrix

5. Display Max Matrix

6. Display Need Matrix

7. Display Available Resources

8. Exit

Enter your choice: 5

Max Matrix:

0 0

0 0

0 0

Menu:

1. Accept Allocation Matrix

2. Accept Max Matrix

3. Accept Available Resources

4. Display Allocation Matrix

5. Display Max Matrix

6. Display Need Matrix

7. Display Available Resources

8. Exit

Enter your choice: 6

Need Matrix:

-3 -1

-2 -1

-2 -3

Menu:

1. Accept Allocation Matrix

2. Accept Max Matrix

3. Accept Available Resources

4. Display Allocation Matrix

5. Display Max Matrix

6. Display Need Matrix

7. Display Available Resources

8. Exit

Enter your choice: 7

Available Resources:

1 2

Menu:

1. Accept Allocation Matrix

2. Accept Max Matrix

3. Accept Available Resources

4. Display Allocation Matrix

5. Display Max Matrix

6. Display Need Matrix

7. Display Available Resources

8. Exit

Enter your choice: 8

**2. Consider the system with 3 resources types A,B, and C with 7,2,6 instancesrespectively.**

**Consider the following snapshot:**

**Answer the following questions:**

**a) Display the contents of Available array?**

**=** Available Resources:

0 0 0

**b) Is there any deadlock? Print the message**

= Deadlock detected: No processes can proceed.

**Ans**

#include <stdio.h>

#define MAX\_PROCESSES 10

#define MAX\_RESOURCES 10

int allot[MAX\_PROCESSES][MAX\_RESOURCES]; // Allocation matrix

int request[MAX\_PROCESSES][MAX\_RESOURCES]; // Request matrix

int totalResources[MAX\_RESOURCES]; // Total resources

int available[MAX\_RESOURCES]; // Available resources

int need[MAX\_PROCESSES][MAX\_RESOURCES]; // Need matrix

int m, n; // Number of processes and resources

// Function to calculate the Need matrix

void calculateNeed() {

for (int i = 0; i < m; i++)

for (int j = 0; j < n; j++)

need[i][j] = request[i][j]; // Here, request is used as need

}

// Function to display the Available array

void displayAvailable() {

printf("Available Resources:\n");

for (int i = 0; i < n; i++) {

printf("%d ", available[i]);

}

printf("\n");

}

// Function to check for deadlock

int isDeadlocked() {

int work[MAX\_RESOURCES];

int finish[MAX\_PROCESSES] = {0};

// Initialize Work with Available resources

for (int i = 0; i < n; i++) {

work[i] = available[i];

}

int count = 0;

while (count < m) {

int found = 0;

for (int p = 0; p < m; p++) {

if (!finish[p]) {

int j;

for (j = 0; j < n; j++)

if (need[p][j] > work[j])

break;

if (j == n) {

// Process can finish

for (int k = 0; k < n; k++)

work[k] += allot[p][k]; // Simulate resource release

finish[p] = 1;

found = 1;

count++;

}

}

}

if (!found) break; // No more processes can proceed

}

// Check if all processes are finished

for (int i = 0; i < m; i++) {

if (!finish[i]) {

return 1; // Deadlock detected

}

}

return 0; // No deadlock

}

int main() {

// Get number of processes and resources from user

printf("Enter the number of processes: ");

scanf("%d", &m);

printf("Enter the number of resources: ");

scanf("%d", &n);

// Input total resources

printf("Enter the total resources (A B C): ");

for (int i = 0; i < n; i++) {

scanf("%d", &totalResources[i]);

}

// Input Allocation Matrix

printf("Enter the Allocation Matrix:\n");

for (int i = 0; i < m; i++) {

printf("P%d: ", i);

for (int j = 0; j < n; j++) {

scanf("%d", &allot[i][j]);

}

}

// Input Request Matrix

printf("Enter the Request Matrix:\n");

for (int i = 0; i < m; i++) {

printf("P%d: ", i);

for (int j = 0; j < n; j++) {

scanf("%d", &request[i][j]);

}

}

// Calculate Available resources

int total\_allocation[MAX\_RESOURCES] = {0};

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) {

total\_allocation[j] += allot[i][j];

}

}

for (int i = 0; i < n; i++) {

available[i] = totalResources[i] - total\_allocation[i];

}

// Display the Available array

displayAvailable();

// Calculate the Need matrix

calculateNeed();

// Check for deadlock

if (isDeadlocked()) {

printf("Deadlock detected: No processes can proceed.\n");

} else {

printf("No deadlock detected.\n");

}

return 0;

}

**Output:**

Enter the number of processes: 5

Enter the number of resources: 3

Enter the total resources (A B C): 7 2 6

Enter the Allocation Matrix:

P0: 0 1 0

P1: 2 0 0

P2: 3 0 3

P3: 2 1 1

P4: 0 0 2

Enter the Request Matrix:

P0: 0 0 0

P1: 2 0 2

P2: 1 0 1

P3: 0 0 2

P4: 0 0 0

Available Resources:

0 0 0

No deadlock detected.

**SET C**

**1. Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.**

**Consider the following snapshot of system, A, B, C and D are the resource type**

**a) Calculate and display the content of need matrix?**

Need Matrix:

0 1 1 -2

-1 1 7 5

2 2 1 1

6 -1 3 0

6 5 1 2

**b) Is the system in safe state? If display the safe sequence.**

System is not in a safe state.

**c) If a request from process P arrives for (0, 4, 2, 0) can it be granted immediately**

**by keeping the system in safe state. Print a message**

Error: Process has exceeded its maximum claim.

**Ans**

#include <stdio.h>

#define MAX\_PROCESSES 10

#define MAX\_RESOURCES 10

int allot[MAX\_PROCESSES][MAX\_RESOURCES];

int max[MAX\_PROCESSES][MAX\_RESOURCES];

int need[MAX\_PROCESSES][MAX\_RESOURCES];

int available[MAX\_RESOURCES];

int m = 5;

int n = 4;

void calculateNeed() {

for (int i = 0; i < m; i++)

for (int j = 0; j < n; j++)

need[i][j] = max[i][j] - allot[i][j];

}

void displayNeed() {

printf("Need Matrix:\n");

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) {

printf("%d ", need[i][j]);

}

printf("\n");

}

}

int isSafe(int safeSeq[]) {

int work[MAX\_RESOURCES];

int finish[MAX\_PROCESSES] = {0};

int count = 0;

for (int i = 0; i < n; i++) {

work[i] = available[i];

}

while (count < m) {

int found = 0;

for (int p = 0; p < m; p++) {

if (!finish[p]) {

int j;

for (j = 0; j < n; j++)

if (need[p][j] > work[j])

break;

if (j == n) {

for (int k = 0; k < n; k++)

work[k] += allot[p][k];

safeSeq[count++] = p;

finish[p] = 1;

found = 1;

}

}

}

if (!found) break;

}

return (count == m);

}

int requestResources(int processNum, int request[]) {

for (int i = 0; i < n; i++) {

if (request[i] > need[processNum][i]) {

printf("Error: Process has exceeded its maximum claim.\n");

return 0;

}

}

for (int i = 0; i < n; i++) {

if (request[i] > available[i]) {

printf("Resources are not available for the request.\n");

return 0;

}

}

for (int i = 0; i < n; i++) {

available[i] -= request[i];

allot[processNum][i] += request[i];

need[processNum][i] -= request[i];

}

int safeSeq[MAX\_PROCESSES];

if (isSafe(safeSeq)) {

printf("Request can be granted. Safe sequence is: ");

for (int i = 0; i < m; i++) {

printf("P%d ", safeSeq[i]);

}

printf("\n");

return 1;

} else {

for (int i = 0; i < n; i++) {

available[i] += request[i];

allot[processNum][i] -= request[i];

need[processNum][i] += request[i];

}

printf("Request cannot be granted as it leads to an unsafe state.\n");

return 0;

}

}

int main() {

int allocation[MAX\_PROCESSES][MAX\_RESOURCES] = {

{0, 0, 1, 2},

{1, 0, 0, 0},

{1, 3, 5, 4},

{0, 6, 3, 2},

{0, 0, 1, 4}

};

int maximum[MAX\_PROCESSES][MAX\_RESOURCES] = {

{0, 1, 2, 0},

{0, 1, 7, 5},

{3, 5, 6, 5},

{6, 5, 6, 2},

{6, 5, 2, 6}

};

available[0] = 1;

available[1] = 5;

available[2] = 2;

available[3] = 0;

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) {

allot[i][j] = allocation[i][j];

max[i][j] = maximum[i][j];

}

}

calculateNeed();

displayNeed();

int safeSeq[MAX\_PROCESSES];

if (isSafe(safeSeq)) {

printf("System is in a safe state. Safe sequence is: ");

for (int i = 0; i < m; i++) {

printf("P%d ", safeSeq[i]);

}

printf("\n");

} else {

printf("System is not in a safe state.\n");

}

int request[] = {0, 4, 2, 0};

requestResources(1, request);

return 0;

}

**Output**

Need Matrix:

0 1 1 -2

-1 1 7 5

2 2 1 1

6 -1 3 0

6 5 1 2

System is not in a safe state.

Error: Process has exceeded its maximum claim.