**Week – 11**

Simulate Bankers Algorithm for Deadlock Avoidance and Deadlock Detection Algorithm

**Code:**

#include <stdio.h>

int np, nr, alloc[100][100], need[100][100], avail[100], total[100][100];

void takeInput() {

printf("Enter no. of processes: ");

scanf("%d", &np);

printf("Enter no. of resources: ");

scanf("%d", &nr);

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

printf("Enter allocated and total resources for process %d\n", i);

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

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

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

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

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

need[i][j] = total[i][j] - alloc[i][j];

}

printf("Enter available resources: ");

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

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

}

void printTable() {

printf("Process \tAlloc\t\tTotal\t\tNeed\n");

char k = 'A';

printf("\t");

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

printf("%c ", k + i);

printf("\t");

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

printf("%c ", k + i);

printf("\t");

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

printf("%c ", k + i);

printf("\n");

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

printf("P%d\t", i);

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

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

printf("\t");

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

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

printf("\t");

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

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

printf("\n");

}

}

int banker() {

int status[100] = {0}, flag, cnt = 0;

int tempAvail[100];

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

tempAvail[i] = avail[i];

}

do {

flag = 0;

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

if (!status[i]) {

int safe = 1;

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

if (tempAvail[j] < need[i][j]) {

safe = 0;

break;

}

}

if (safe) {

flag = 1;

status[i] = 1;

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

tempAvail[j] += alloc[i][j];

cnt++;

}

}

}

} while (cnt < np && flag);

if (cnt == np) {

printf("Safe sequence exists.\n");

return 1;

} else {

printf("Unsafe state detected.\n");

return 0;

}

}

void deadlockDetection() {

int work[100], finish[100] = {0}, i, j;

for (i = 0; i < nr; i++) {

work[i] = avail[i];

}

int deadlockProcesses[100], deadlockCount = 0;

for (i = 0; i < np; i++) {

int canFinish = 0;

if (!finish[i]) {

for (j = 0; j < nr; j++) {

if (need[i][j] > work[j]) {

canFinish = 0;

break;

} else {

canFinish = 1;

}

}

if (canFinish) {

for (j = 0; j < nr; j++) {

work[j] += alloc[i][j];

}

finish[i] = 1;

i = -1;

}

}

}

for (i = 0; i < np; i++) {

if (!finish[i]) {

deadlockProcesses[deadlockCount++] = i;

}

}

if (deadlockCount > 0) {

printf("Processes involved in the deadlock cycle: ");

for (i = 0; i < deadlockCount; i++) {

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

}

printf("\n");

} else {

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

}

}

void handleExtraRequest() {

int process, extraRequest[100];

printf("Enter the process number for the extra request: ");

scanf("%d", &process);

printf("Enter the additional request for each resource:\n");

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

printf("Resource %c: ", 'A' + i);

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

}

int canGrant = 1;

if (!canGrant) {

printf("The request cannot be granted.\n");

return;

}

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

need[process][i] += extraRequest[i];

alloc[process][i] -= extraRequest[i];

}

if (banker()) {

printf("The request can be granted. Checking for deadlock...\n");

deadlockDetection();

} else {

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

// Revert the need matrix and available resources

deadlockDetection();

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

need[process][i] -= extraRequest[i];

alloc[process][i] += extraRequest[i];

}

}

}

int main() {

takeInput();

printTable();

// Check initial safety

if (!banker()) {

printf("Initial state is unsafe. Exiting.\n");

return 0;

}

int y=1;

// Loop to handle extra requests and check for deadlocks

while(y==1) {

handleExtraRequest();

printf("want to continue 1/0");

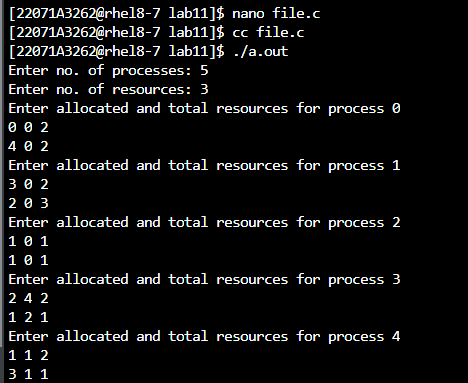
scanf("%d",&y);

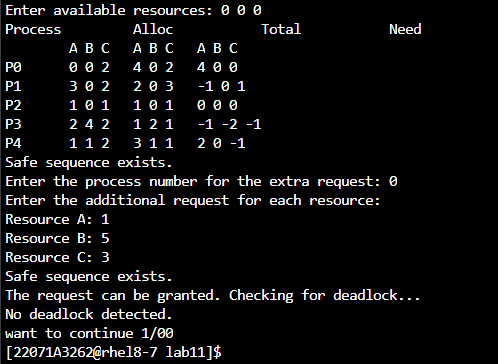
}

return 0;

}

**Output :**

****

****