|  |
| --- |
| **Task 1: Determine safe/unsafe situation using Deadlock detection algorithm** |

#include <stdio.h>

int main()

{

int n, m, i, j, k;

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

scanf("%d", &n);

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

scanf("%d", &m);

int alloc[n][m], max[n][m], need[n][m], avail[m], finish[n], safe\_seq[n], count = 0;

// Input allocation matrix

printf("Enter the allocation matrix: \n");

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

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

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

}

}

// Input maximum matrix

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

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

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

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

}

}

// Calculate need matrix

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

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

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

}

}

// Input available matrix

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

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

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

}

// Initialize finish array as false for each process

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

finish[i] = 0;

}

// Banker's Algorithm

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

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

if(finish[i] == 0) {

int flag = 1;

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

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

flag = 0;

break;

}

}

if(flag) {

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

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

}

finish[i] = 1;

safe\_seq[count++] = i;

}

}

}

}

// Check for deadlock

int deadlock = 0;

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

if(finish[i] == 0) {

printf("Process %d is deadlocked.\n", i);

deadlock = 1;

}

}

if(!deadlock) {

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

printf("Safe sequence is: ");

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

printf("%d ", safe\_seq[i]);

}

printf("\n");

}

return 0;

}

|  |
| --- |
| **Task 2: fixed partition in memory management** |

#include<stdio.h>

main()

{

int ms, bs, nob, ef,n,

mp[10],tif=0; int i,p=0;

printf("Enter the total memory available (in Bytes) -- ");

scanf("%d",&ms);

printf("Enter the block size (in Bytes) -- ");

scanf("%d", &bs);

nob=ms/bs;

ef=ms - nob\*bs;

printf("\nEnter the number of processes -- ");

scanf("%d",&n);

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

{

printf("Enter memory required for process %d (in Bytes)-- ",i+1);

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

}

printf("\nNo. of Blocks available in memory--%d",nob);

printf("\n\n PROCESS\t MEMORYREQUIRED \t ALLOCATED \t INTERNAL FRAGMENTATION");

for(i=0;i<n && p<nob;i++)

{

printf("\n %d\t\t%d",i+1,mp[i]);

if(mp[i] > bs)

printf("\t\tNO\t\t---");

else

{

printf("\t\tYES\t%d",bs-mp[i]);

tif = tif + bs-mp[i];

p++;

}

}

if(i<n)

printf("\nMemory is Full, Remaining Processes cannot be accomodated");

printf("\n\nTotal Internal Fragmentation is %d",tif);

printf("\nTotal External Fragmentation is %d",ef);

}