Bahria University

Karachi Campus



LAB EXPERIMENT NO.

11

LIST OF TASKS

|  |  |
| --- | --- |
| **TASK NO** | **OBJECTIVE** |
| 1 | Write a short note on Banker’s algorithm stating its main purpose and working mechanism. |
| 2 | Implement the Banker’s Algorithm explained above in C language. |
|  |  |
|  |  |

Submitted On:

**08-06-2023**

(Date: DD/MM/YY)

**Task # 01:**

**Solution:**

The Banker's algorithm is a resource allocation and deadlock avoidance algorithm developed by Edsger Dijkstra. It is used to ensure that a system will not enter a deadlock state. The algorithm works by first calculating the maximum number of resources that each process can use. It then calculates the number of resources that are currently available. Finally, it checks if the system is in a safe state. If the system is in a safe state, then the algorithm will allow the processes to continue using resources. If the system is not in a safe state, then the algorithm will block the processes from using resources.

The main purpose of the Banker's algorithm is to prevent deadlocks. A deadlock occurs when a process is waiting for a resource that is being held by another process, which is in turn waiting for a resource that is being held by the first process. The Banker's algorithm prevents deadlocks by ensuring that no process can hold more resources than it needs.

The Banker's algorithm works by first calculating the maximum amount of resources that each process can use. This is done by creating a matrix where each row represents a process and each column represents a resource. The values in the matrix represent the maximum amount of each resource that each process can use.

**Task # 01:**

**Solution:**

#include <stdio.h>

#define MAX\_RESOURCES 10

#define MAX\_PROCESSES 10

the available resources

int checkSafe(int need[][MAX\_RESOURCES], int available[], int n, int m, int allocated[][MAX\_RESOURCES]) {

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

int i, j, k;

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

work[i] = available[i];

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

if (finish[i] == 0) {

int canAllocate = 1;

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

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

canAllocate = 0;

break;

}

}

if (canAllocate) {

// Allocating resources to the current process

for (k = 0; k < m; k++)

work[k] += allocated[i][k];

finish[i] = 1;

i = -1; // Start checking from the beginning as resources might be released

}

}

}

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

if (finish[i] == 0)

return 0; // Not safe

}

}

int main() {

int available[MAX\_RESOURCES], allocated[MAX\_PROCESSES][MAX\_RESOURCES], need[MAX\_PROCESSES][MAX\_RESOURCES];

int n, m; // n = number of processes, m = number of resources

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

scanf("%d", &n);

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

scanf("%d", &m);

printf("Enter the available resources: ");

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

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

printf("Enter the allocated resources for each process:\n");

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

printf("Process %d: ", i + 1);

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

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

}

printf("Enter the maximum resources needed for each process:\n");

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

printf("Process %d: ", i + 1);

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

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

}

if (checkSafe(need, available, n, m, allocated))

printf("Safe state: Resources can be allocated to all processes.\n");

else

printf("Unsafe state: Resources cannot be allocated to all processes.\n");}

**OUTPUT:**



