Bahria University,

Karachi Campus



LAB EXPERIMENT NO.

\_\_\_11\_\_\_

LIST OF TASKS

|  |  |
| --- | --- |
| TASK NO | OBJECTIVE |
| **01** | Write a short note on Banker’s algorithm stating its main purpose and working mechanism. |
| **02** | Implement the Banker’s Algorithm explained above in C language. |
|  |  |
|  |  |

Submitted On

06-04-2023

(Date: DD/MM/YY)

**Task 01:** Write a short note on Banker’s algorithm stating its main purpose and working mechanism.

**Solution:**

The Banker's algorithm is a resource allocation and deadlock avoidance algorithm used in operating systems. Its main purpose is to ensure the safe and efficient allocation of resources to different processes, while also preventing deadlocks.

The Banker's algorithm works by considering the available resources in a system and the maximum resources that each process may need to complete its execution. It uses this information to determine if granting a resource request will result in a safe state, i.e., a state where all processes can eventually complete without encountering a deadlock.

**The algorithm operates as follows:**

1. Initialization: The system must first determine the total number of resources available in each resource category and keep track of the allocated and maximum resource needs of each process.

2. Request Evaluation: When a process makes a request for additional resources, the Banker's algorithm checks if granting the request will result in a safe state. It examines whether the system has enough available resources to satisfy the request and whether granting the request will lead to a sequence of resource allocations that can avoid a deadlock.

3. Resource Allocation: If the requested resources can be safely allocated, the system grants the resources to the requesting process. It updates the available resources and marks the allocated resources as in use by the process.

4. Resource Release: When a process finishes using its allocated resources, it releases them back to the system, making them available for allocation to other processes.

5. Deadlock Avoidance: The Banker's algorithm ensures that resource allocation does not lead to a deadlock situation. It analyzes the resource requests and resource allocation history to ensure that there is always a safe sequence of resource allocations, preventing the occurrence of deadlocks.

By using the Banker's algorithm, the operating system can make informed decisions about resource allocation, avoiding scenarios where processes may be left in a deadlock state or resources are overcommitted. This helps to enhance system stability and efficiency by preventing resource deadlocks and ensuring that all processes can execute to completion.

**Task 02:** Implement the Banker’s Algorithm explained above in C language.

**Solution:**

#include <stdio.h>

#include <stdlib.h

#define MAX\_PROCESSES 10

#define MAX\_RESOURCES 10

int available[MAX\_RESOURCES];

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

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

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

int work[MAX\_RESOURCES];

int finish[MAX\_PROCESSES];

int num\_processes, num\_resources;

void read\_input() {

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 (int i = 0; i < num\_resources; i++) {

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

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

for (int i = 0; i < num\_processes; i++) {

for (int j = 0; j < num\_resources; j++) {

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

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

for (int i = 0; i < num\_processes; i++) {

for (int j = 0; j < num\_resources; j++) {

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

need[i][j] = maximum[i][j] - allocation[i][j];}}}

int safety\_check() {

int i, j, k;

int count = 0;

for (i = 0; i < num\_resources; i++) {

work[i] = available[i];}

for (i = 0; i < num\_processes; i++) {

finish[i] = 0;}

while (count < num\_processes) {

int found = 0;

for (i = 0; i < num\_processes; i++) {

if (finish[i] == 0) {

int can\_finish = 1;

for (j = 0; j < num\_resources; j++) {

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

can\_finish = 0;

break;}}

if (can\_finish) {

for (k = 0; k < num\_resources; k++) {

work[k] += allocation[i][k];}

finish[i] = 1;

found = 1;

count++;}}}

if (!found) {

return 0;}}

return 1;}

int request\_resources(int process\_id, int request[]) {

int i;

for (i = 0; i < num\_resources; i++) {

if (request[i] > need[process\_id][i]) {

return -1;}

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

return -1;}}

for (i = 0; i < num\_resources; i++) {

available[i] -= request[i];

allocation[process\_id][i] += request[i];

need[process\_id][i] -= request[i];}

if (safety\_check()) {

return 0;

} else {

for (i = 0; i < num\_resources; i++) {

available[i] += request[i];

allocation[process\_id][i] -= request[i];

need[process\_id][i] += request[i];}

return -1;}}

int main() {

int process\_id, request[MAX\_RESOURCES];

read\_input();

if (safety\_check()) {

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

} else {

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

exit(0);}

printf("Enter the process id to request resources: ");

scanf("%d", &process\_id);

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

for (int i = 0; i < num\_resources; i++) {

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

if (request\_resources(process\_id, request) == 0) {

printf("Request granted.\n");

} else {

printf("Request denied.\n");}

return 0;}

**Output:**

