Name ⇒ Ayush. D. C
Roll no ⇒ 04
PRN ⇒ 1032231720.
TY CSE D1

~~OAIES~~ OS

ASSIGNMENT : 07

* Problem Statement:
Write an program to simulate Banker's Algorithm for deadlock avoidance.

* Objective:

- To understand the Concept of Deadlock in Os
- To study the Banker's algorithm as method of deadlock avoidance
- To implement a solution that ensures safe state resource allocation

* Theory:
The Banker's Algorithm is a deadlock avoidance algorithm used in Os to allocate resources the processed in a way that ensures the system retains in safe state. It is named after a banking system analogy where banker ensures that loans are allocated where bunker ensures that loans are allocated in such a way that they can always be repaid.

* Key Concepts:

1) Safe state: A state where system can allocate resources to all process in some order without leading to deadlock.
2) Unsafe State: A state where resources allocation might lead to Deadlock

* Algorithm steps:

→ Resource - Request Algorithm :
Request ; = request vector for process P_i
If Request$_i$[j] = k then process P_i wants k instances of resource type R_j

1. If Request$_i$ ≤ Need$_i$ go to step 2. Otherwise raise error since, process has exceeded its maximum claim
2. If Request$_i$ ≤ Available, go to step 3. Otherwise, P_i must wait, since resources are not available.
3. Pretend to allocate requested resources to P_i by modifying the state as follows:
available = Available − Request.
Allocation = Allocation + Request.
Need = Need − Request.

If safe ⇒ Resources must be Allocated to P_i
If unsafe ⇒ P_i must wait and old resource-allocation state restored.

→ Safety Algorithm:
To determine whether system is in safe state
1. let work and finish be vectors of length m and n respectively
Intialize: Work := Available
finish[i] = True for all i, then system is in safe state.

* Conclusion: Thus, we have succesfully implemented Banker's Algorithm to simulate safe resource allocation and avoid deadlocks. This algorithm helps ensure that the system never enters an unsafe state, making it a vital strategy in OS resource management.

FAQ's:

1. What are key components of Bank algorithm?

Bankers Algorithm is used for deadlocks avoidance in Os. It works by checking whether granting a resource request will leave the system in safe state.

Key components:

a). Processes (n) ⇒ set of ~~resource~~ process competing for resources.

b) Resource types (m) ⇒ set of resource types available in system.

c) Data structures ⇒

- available [m]: Indicates no. of available instances of each resource
- Max [n][m]
- Allocation [n][m]:
- need [n][m]:

Advantages & Disadvantages of Banker Algorithm.

Advantages:

- Deadlock Avoidance
- Optimal Resource Utilization
- Predictable & systematic
- Flexible with multiple resources.

Disadvantages:

- Requires Prior Knowledge
- ~~Costly~~ Complexity.
- Reduced throughout
- Not suitable for Dynamic System
- Overhead.

3) Which process will finish last?

	Allocation			Request		
	X	Y	Z	X	Y	Z
P0	2	1	1	1	0	3
P1	2	0	1	0	1	2
P2	2	2	1	1	2	0

→

step 1: Available = Total − sum (Allocation)

sum Alloc X = 5 → Available X = 5−5 = 0

sum Alloc Y = 4 → Available Y = 5−4 = 1

sum Alloc Z = 3 → Available Z = 5−3 = 2

so available = (0, 1, 2)

step 2: Process whose Request ≤ Available

- P0 request (1, 0, 3) : X : 1 > 0 → cannot run
- P1 request (0, 1, 2) = 0 ≤ 0, 1 ≤ 1, 2 ≤ 2 → P
- → P2 request (1, 2, 0) : Y : 2 > 1 → cannot run

Available = (0, 1, 2) + (2, 0, 1) = (2, 1, 3)

step 3: - P0 request (1, 0, 3) : 1 ≤ 2, 0 ≤ 1, 3 ≤ 3

- P2 request (1, 2, 0) : Y : 2 > 1 → cannot yet.

let P0 finish & release (1, 2, 1)

∴ Available = (2, 1, 2) + (1, 2, 1) = (3, 3, 4)

step 4: - P2 request (1, 2, 0) : 1 ≤ 3, 2 ≤ 3, 0

P2 can now

Safe seq found : P1 → P0 → P2

∴ LAST finished process is P2.

* Input

Enter no of process and no of resource : 5 3
Enter the available resource : 3 3 2
Enter the MAX matrix :
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3

Enter Allocation matrix :
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2

Available resources :
3 3 2

* Output :

Need Matrix :
7 4 3
1 2 2
6 0 0
0 1 1
4 3 1

system in safe state

Safe sequence : P1 → P3 → P4 → P0 → P2

Enter Process number for Request : 1
Enter request Vector : 1 0 2
Request can be granted.
So new safe sequence : P1 → P3 → P4 → P0 → P2

Code:

```
include <stdio.h>
#include <stdbool.h>

#define MAX_PROCESSES 10
#define MAX_RESOURCES 10

int main()
{
    int n, m; // n = number of processes, m = number of resources
    int alloc[MAX_PROCESSES][MAX_RESOURCES];
    int max[MAX_PROCESSES][MAX_RESOURCES];
    int need[MAX_PROCESSES][MAX_RESOURCES];
    int avail[MAX_RESOURCES];
    bool finish[MAX_PROCESSES];

    printf("Enter the number of processes: ");
    scanf("%d", &n);
    printf("Enter the number of resources: ");
    scanf("%d", &m);

    printf("Enter the allocation matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            scanf("%d", &alloc[i][j]);

    printf("Enter the maximum matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            scanf("%d", &max[i][j]);

    printf("Enter the available resources:\n");
    for (int i = 0; i < m; i++)
        scanf("%d", &avail[i]);

    // Calculate need matrix
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            need[i][j] = max[i][j] - alloc[i][j];

    // Print the need matrix
    printf("\nNeed Matrix:\n");
    for (int i = 0; i < n; i++)
```

```
{
    printf("P%d: ", i);
    for (int j = 0; j < m; j++)
    {
        printf("%d ", need[i][j]);
    }
    printf("\n");
}

// Initialize finish array
for (int i = 0; i < n; i++)
    finish[i] = false;

int safeSeq[MAX_PROCESSES];
int work[MAX_RESOURCES];
for (int i = 0; i < m; i++)
    work[i] = avail[i];

int count = 0;
bool found;

// Initial safety check to find safe sequence
while (count < n)
{
    found = false;
    for (int i = 0; i < n; i++)
    {
        if (!finish[i])
        {
            bool canAllocate = true;
            for (int j = 0; j < m; j++)
            {
                if (need[i][j] > work[j])
                {
                    canAllocate = false;
                    break;
                }
            }
            if (canAllocate)
            {
                for (int j = 0; j < m; j++)
                    work[j] += alloc[i][j];

                safeSeq[count++] = i;
```

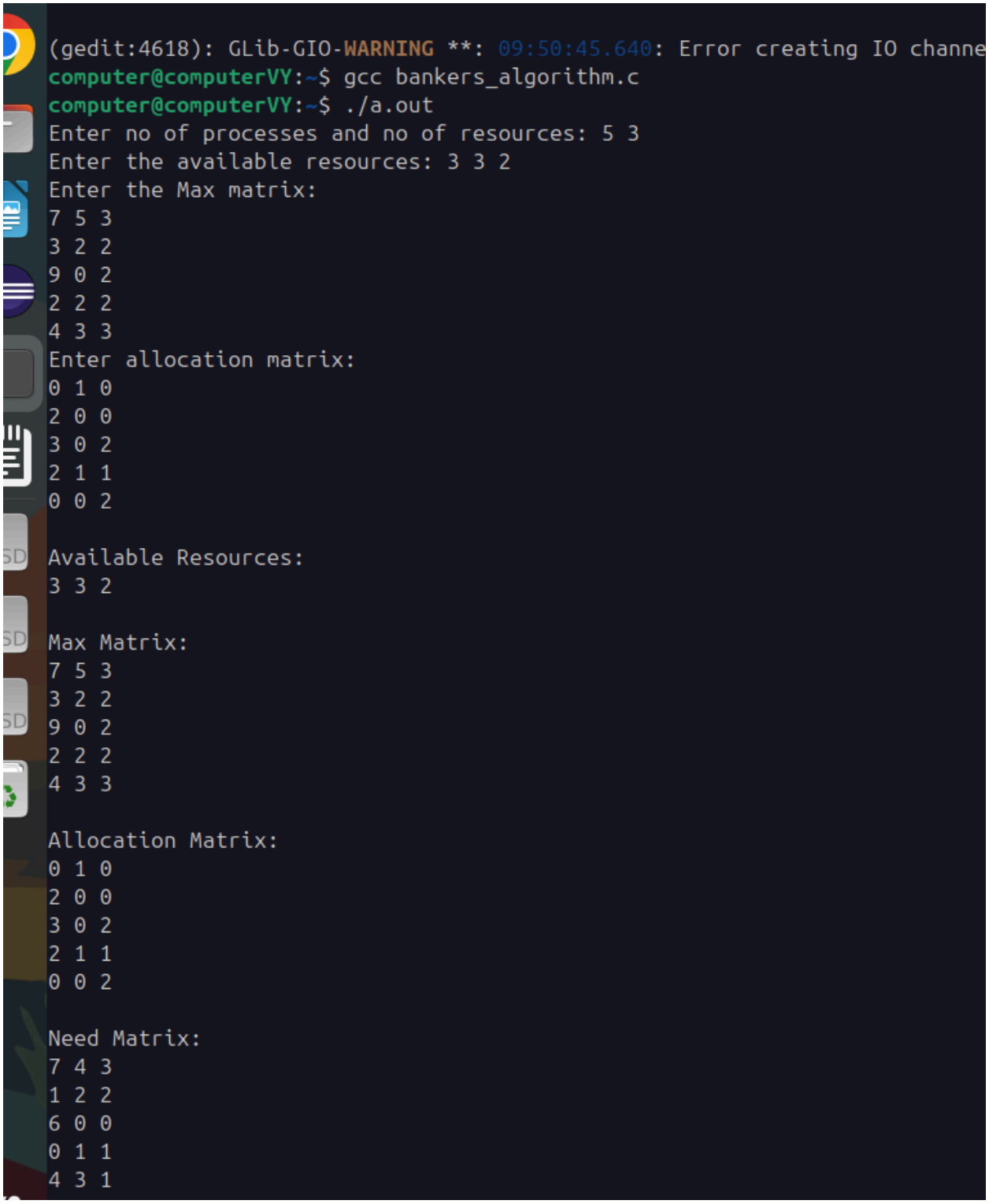
```
                finish[i] = true;
                found = true;
            }
        }
    }

    if (!found)
    {
        printf("\nSystem is NOT in a safe state (deadlock is possible).\n");
        return 0;
    }
}

// If system is safe
printf("\nSystem is in a SAFE state.\nSafe Sequence: ");
for (int i = 0; i < n; i++)
    printf("P%d ", safeSeq[i]);
printf("\n");

// Now handle requests repeatedly
while (true) {
    int reqProcess;
    printf("\nEnter process number for request (-1 to exit): ");
    scanf("%d", &reqProcess);

    if (reqProcess == -1) {
        printf("Exiting request loop.\n");
        break;
    }

    if (reqProcess < 0 || reqProcess >= n) {
        printf("Invalid process number.\n");
        continue;
    }

    int request[MAX_RESOURCES];
    printf("Enter request vector: ");
    for (int i = 0; i < m; i++)
        scanf("%d", &request[i]);

    // Check if request <= need
    bool validRequest = true;
    for (int i = 0; i < m; i++) {
        if (request[i] > need[reqProcess][i]) {
```

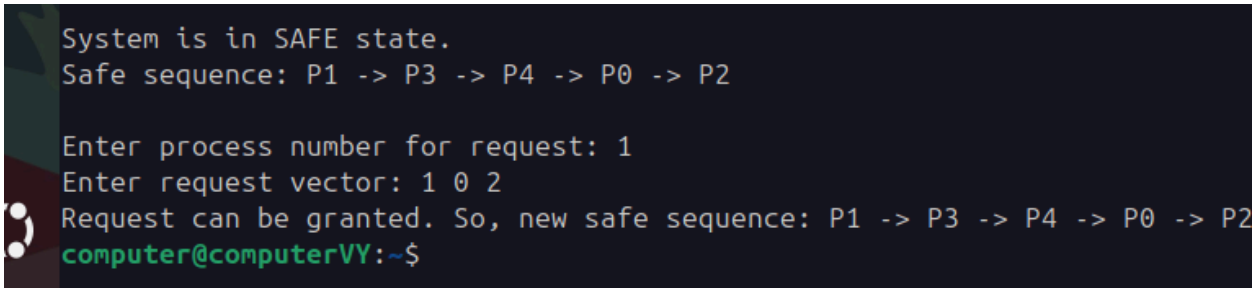
```
            printf("Error: Request exceeds the process's maximum need.\n");
            validRequest = false;
            break;
        }
    }
    if (!validRequest) continue;

    // Check if request <= available
    for (int i = 0; i < m; i++) {
        if (request[i] > avail[i]) {
            printf("Request cannot be granted immediately due to insufficient available
resources.\n");
            validRequest = false;
            break;
        }
    }
    if (!validRequest) continue;

    // Pretend to allocate requested resources
    for (int i = 0; i < m; i++) {
        avail[i] -= request[i];
        alloc[reqProcess][i] += request[i];
        need[reqProcess][i] -= request[i];
    }

    // Re-run safety check after allocation
    for (int i = 0; i < m; i++)
        work[i] = avail[i];
    for (int i = 0; i < n; i++)
        finish[i] = false;

    count = 0;
    bool foundSeq = true;

    while (count < n) {
        bool found = false;
        for (int i = 0; i < n; i++) {
            if (!finish[i]) {
                bool canAllocate = true;
                for (int j = 0; j < m; j++) {
                    if (need[i][j] > work[j]) {
                        canAllocate = false;
                        break;
                    }
```

```
                }
                if (canAllocate) {
                    for (int j = 0; j < m; j++)
                        work[j] += alloc[i][j];
                    safeSeq[count++] = i;
                    finish[i] = true;
                    found = true;
                }
            }
        }
        if (!found) {
            foundSeq = false;
            break;
        }
    }

    if (!foundSeq) {
        // Rollback allocation
        for (int i = 0; i < m; i++) {
            avail[i] += request[i];
            alloc[reqProcess][i] -= request[i];
            need[reqProcess][i] += request[i];
        }
        printf("Request cannot be granted as it leads to unsafe state.\n");
    } else {
        printf("Request can be granted. So, new safe sequence: ");
        for (int i = 0; i < n; i++) {
            printf("P%d", safeSeq[i]);
            if (i != n - 1) printf(" -> ");
        }
        printf("\n");
    }
  }

  return 0;
}
```

OUTPUT:

ON NEXT PAGE:


(gedit:4618): GLib-GIO-WARNING **: 09:50:45.640: Error creating IO channe
computer@computerVY:~$ gcc bankers_algorithm.c
computer@computerVY:~$ ./a.out
Enter no of processes and no of resources: 5 3
Enter the available resources: 3 3 2
Enter the Max matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter allocation matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Available Resources:
3 3 2
Max Matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Allocation Matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Need Matrix:
7 4 3
1 2 2
6 0 0
0 1 1
4 3 1


System is in SAFE state.
Safe sequence: P1 -> P3 -> P4 -> P0 -> P2
Enter process number for request: 1
Enter request vector: 1 0 2
Request can be granted. So, new safe sequence: P1 -> P3 -> P4 -> P0 -> P2
computer@computerVY:~$