

College Of Engineering Trivandrum

System Software Lab



Mohammed Rabeeh
S5 CSE Roll No:35

TVE18CS036

Department of Computer Science

September 14, 2020

Contents

1	Aim	2
2	Algorithms	2
2.1	Safety Algorithm	2
2.2	Resource Request Algorithm	2
3	Program Code	2
4	Program Output	4
5	Result	4



Cycle 1

Exp No 4

1 Aim

Implement the banker's algorithm for deadlock avoidance.

2 Algorithms

2.1 Safety Algorithm

1. Let Work and Finish be vectors of length 'm' and 'n' respectively. Initialize: Work = Available Finish[i] = false; for i=1, 2, 3, 4...n
2. Find an i such that both
 - (a) Finish[i] = false
 - (b) Need[i] less than or equal to Workif no such i exists go to Step 4.
3. Work = Work + Allocation[i] Finish[i] = true Go to Step 2.
4. if Finish [i] = true for all i then the system is in a safe state

2.2 Resource Request Algorithm

1. If Request[i] is less than to Need[i], Go to Step 2 ; Otherwise, raise an error condition, since the process has exceeded its maximum claim.
2. If Request[i] is greater than or equal to Available Go to Step (3); Otherwise, P[i] must wait, since the resources are not available.
3. Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows:

```
Available[i] = Available[i] - Request[i]
Allocation[i] = Allocation[i] + Request[i]
Need[i] = Need[i] - Request[i]
```

3 Program Code

```
#include <stdio.h>

void readMatrix(int arr[50][50], int n, int m) {
    int i,j;
    for(i = 0; i < n; i++) {
        printf("P[%d]: ", i);
        for(j = 0; j < m; j++) {
            scanf("%d", &arr[i][j]);
        }
    }
}
```

```

int main()
{

int n, m, i, j, k, q, alloc[50][50], max[50][50], avail[50];
n = 5;
m = 3;
printf("Enter number of processes: ");
scanf("%d", &n);
printf("Enter number of resources: ");
scanf("%d", &m);
printf("\nEnter the allocation matrix:\n");
readMatrix(alloc, n ,m);
printf("\nEnter the max matrix:\n");
readMatrix(max, n ,m);
printf("\nEnter the available resources: ");
for(q = 0; q < m; q++) {
    scanf("%d", &avail[q]);
}

int f[n], ans[n], ind = 0, flag;
for (k = 0; k < n; k++) {
f[k] = 0;
}
int need[n][m];
for (i = 0; i < n; i++) {
for (j = 0; j < m; j++)
need[i][j] = max[i][j] - alloc[i][j];
}
int y = 0;
for (k = 0; k < 5; k++) {
for (i = 0; i < n; i++) {
if (f[i] == 0) {

flag = 0;
for (j = 0; j < m; j++) {
if (need[i][j] > avail[j]){
flag = 1;
break;
}
}

if (flag == 0) {
ans[ind++] = i;
for (y = 0; y < m; y++)
avail[y] += alloc[i][y];
f[i] = 1;
}
}
}
}

if(flag) {
printf("NOT IN SAFE STATE\n");
} else {
printf("Following is a Safe Sequence\n");
for (i = 0; i < n - 1; i++)
printf(" P%d ->", ans[i]);
printf(" P%d\n", ans[n - 1]);
}

return (0);
}

```

4 Program Output

Input

Number of Processes = 5
Number of resources = 3

Allocation Matrix:

	A	B	C
P[0]:	0	1	0
P[1]:	2	0	0
P[2]:	3	0	2
P[3]:	2	1	1
P[4]:	0	0	2

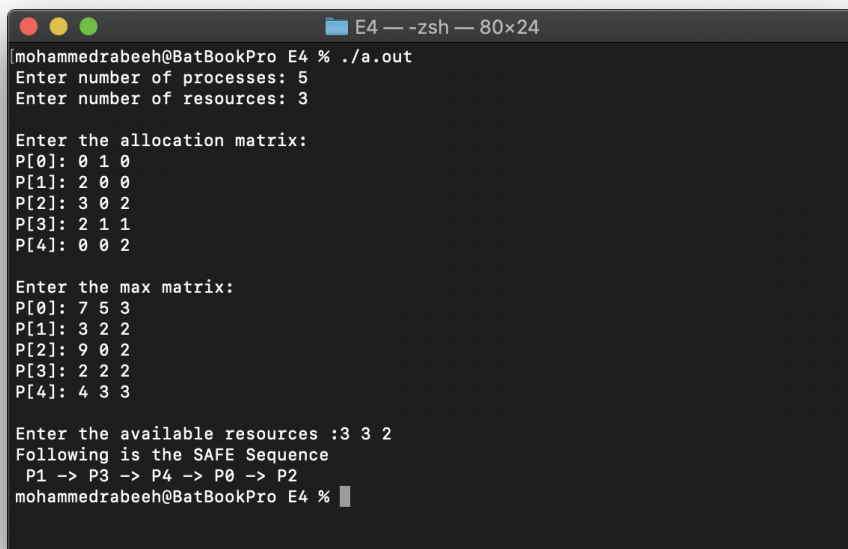
Max Matrix:

	A	B	C
P[0]:	7	5	3
P[1]:	3	2	2
P[2]:	9	0	2
P[3]:	2	2	2
P[4]:	4	3	3

Available:

A	B	C
3	3	2

Output



```
mohammedrabeeh@BatBookPro E4 % ./a.out
Enter number of processes: 5
Enter number of resources: 3

Enter the allocation matrix:
P[0]: 0 1 0
P[1]: 2 0 0
P[2]: 3 0 2
P[3]: 2 1 1
P[4]: 0 0 2

Enter the max matrix:
P[0]: 7 5 3
P[1]: 3 2 2
P[2]: 9 0 2
P[3]: 2 2 2
P[4]: 4 3 3

Enter the available resources :3 3 2
Following is the SAFE Sequence
P1 -> P3 -> P4 -> P0 -> P2
mohammedrabeeh@BatBookPro E4 %
```

5 Result

Banker's Algorithm was implemented in C Language and their Output verified. The program was executed on macOS Catalina 10.15.3 operating system using ZSH Shell.