Department of
Computer Science and Engineering

Title: Banker's Algorithm and Deadlock Avoidance

Operating System Lab

CSE 310

Green University of Bangladesh

1 Objective(s)

- To gather knowledge of Banker's algorithm and how it avoids deadlock.
- To implement Bankers algorithm to check whether a system will be in safe state or not.
- To implement the code for getting a safe sequence of processes from Banker's algorithm.
- To implement the code for requesting additional resources from Banker's algorithm.

2 Problem analysis

Banker's algorithm is used to avoid deadlock and allocate resources safely to each process in the computer system. The 'S-State' examines all possible tests or activities before deciding whether the allocation should be allowed to each process. It also helps the operating system to successfully share the resources between all the processes. When a new process is created in a computer system, the process must provide all types of information to the operating system like upcoming processes, requests for their resources, counting them, and delays. Based on these criteria, the operating system decides which process sequence should be executed or waited so that no deadlock occurs in a system.

2.1 Banker's Algorithm for Checking whether Resources can be Allocated Safely

Algorithm 1: Safety Algorithm

Result: State whether it is Safe of Unsafe for the given list of processes

```
while There remains a process which is unfinished do
    if Need ≤ Available then
        Execute Process;
        New Available = Available + Allocation;
    else
        Do not execute;
        Go to the next process;
```

3 Implementation in C

```c
#include <stdio.h>
#include <stdlib.h>


void takeInputAndCalculateNeedMatrix(int A[][10], int N[][10], int M[10][10],
    int W[1][10], int *n, int *m)
{
    int i, j;
    printf("\n Enter total number of processes : ");
    scanf("%d", n);
    printf("\n Enter total number of resources : ");
    scanf("%d", m);
    for (i = 0; i < *n; i++)
    {
        printf("\n Process %d\n", i + 1);
        for (j = 0; j < *m; j++)
        {
            printf(" Allocation for resource %d : ", j + 1);
            scanf("%d", &A[i][j]);
            printf(" Maximum for resource %d : ", j + 1);
            scanf("%d", &M[i][j]);
        }
    }
```

```
23      printf("\n Available resources : \n");
24      for (i = 0; i < *m; i++)
25      {
26          printf(" Resource %d : ", i + 1);
27          scanf("%d", &W[0][i]);
28      }
29
30      for (i = 0; i < *n; i++)
31          for (j = 0; j < *m; j++)
32              N[i][j] = M[i][j] - A[i][j];
33
34  }
35
36  int safety(int A[][10], int N[][10], int B[1][10], int n, int m)
37  {
38
39      int i, j, k, x = 0, f1 = 0, f2 = 0;
40      int F[10], W[1][10];
41      for (i = 0; i < n; i++)
42          F[i] = 0;
43      for (i = 0; i < m; i++)
44          W[0][i] = B[0][i];
45
46      for (k = 0; k < n; k++)
47      {
48          for (i = 0; i < n; i++)
49          {
50              if (F[i] == 0)
51              {
52                  f2 = 0;
53                  for (j = 0; j < m; j++)
54                  {
55                      if (N[i][j] > W[0][j])
56                          f2 = 1;
57                  }
58                  if (f2 == 0 && F[i] == 0)
59                  {
60                      for (j = 0; j < m; j++)
61                          W[0][j] += A[i][j];
62                      F[i] = 1;
63                      f1++;
64                  }
65              }
66          }
67          if (f1 == n)
68              return 1;
69      }
70      return 0;
71  }
72
73
74  int banker(int A[][10], int N[][10], int W[1][10], int n, int m)
75  {
76      int j, i;
77      j = safety(A, N, W, n, m);
78      if (j != 0)
79      {
80          printf("\n\nIt safe to allocate resources.\n");
```

```
81          return 1;
82      }
83      else
84      {
85          printf("\n Deadlock has occured.\n");
86          return 0;
87      }
88  }
89
90  int main()
91  {
92      int All[10][10], Max[10][10], Need[10][10]
93      , W[1][10];
94      int n, m, pid, c, r;
95
96      takeInputAndCalculateNeedMatrix(All, Need, Max, W, &n, &m);
97      banker(All, Need, W, n, m);
98
99      return 0;
100 }
```

4 Input/Output

4.1 Input

Input of the program is given below.

```
Enter total number of processes : 5
Enter total number of resources : 3

Process 1
Allocation for resource 1 : 0    Maximum for resource 1 : 7
Allocation for resource 2 : 1    Maximum for resource 2 : 5
Allocation for resource 3 : 0    Maximum for resource 3 : 3

Process 2
Allocation for resource 1 : 2    Maximum for resource 1 : 3
Allocation for resource 2 : 0    Maximum for resource 2 : 2
Allocation for resource 3 : 0    Maximum for resource 3 : 2

Process 3
Allocation for resource 1 : 3    Maximum for resource 1 : 9
Allocation for resource 2 : 0    Maximum for resource 2 : 0
Allocation for resource 3 : 2    Maximum for resource 3 : 2

Process 4
Allocation for resource 1 : 2    Maximum for resource 1 : 2
Allocation for resource 2 : 1    Maximum for resource 2 : 2
Allocation for resource 3 : 1    Maximum for resource 3 : 2

Process 5
Allocation for resource 1 : 0    Maximum for resource 1 : 4
Allocation for resource 2 : 0    Maximum for resource 2 : 3
Allocation for resource 3 : 2    Maximum for resource 3 : 3

Available resources :
Resource 1 : 3
Resource 2 : 3
Resource 3 : 2
```

4.2 Output

Output of the program is given below.

```
It safe to allocate resources.
```

5 Discussion & Conclusion

Based on the focused objective(s) to understand about Banker's algorithm determining whether it is safe to allocate resources, the additional lab exercise made me more confident towards the fulfilment of the objectives(s).

6 Lab Task (Please implement yourself and show the output to the instructor)

1. To implement the code for getting a safe sequence of processes from Banker's algorithm.

6.1 Problem analysis

Study and analyze the code given in the section before. Write code for getting the safe sequence of processes from the Banker's algorithm.

6.2 Input

Input of the program is given below.

```
Use the input given in 4.1.
```

6.3 Output

Output of the program is given below.

```
A safety sequence has been detected.
P1 P3 P4 P0 P2
```

7 Lab Exercise (Submit as a report)

- To implement the code for requesting additional resources from Banker's algorithm which generates safe sequence of processes.

Algorithm 2: Additional Resource Request Algorithm

Result: After Requesting for Additional Resources State Whether it is Safe of Unsafe for the Given List of Processes

```
1  if Request ≤ Need then
2  |  Go to line 5;
3  else
4  |  Error;
5  if Request ≤ Available then
6  |  Go to line 9;
7  else
8  |  Wait;
9  Available = Available - Request;
10 Allocation = Allocation + Request;
11 Need = Need - Request;
12 Check if new state is safe or not.
13
```

7.1 Input

Input of the program is given below.

```
First, use the input given in 4.1.
Then use the inputs given below.

Do you want make an additional request for any of the process ? (1=Yes|0=No): 1

Enter process number : 2

Enter additional request :
Request for resource 1 : 1
Request for resource 2 : 0
Request for resource 3 : 2
```

7.2 Output

Output of the program is given below.

```
A safety sequence has been detected after granting additional request.
P1 P3 P4 P0 P2
```

8 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.