Assignment No 4:

Name: Samruddhi Devram Khilari

Branch & Year: AIML-B SY

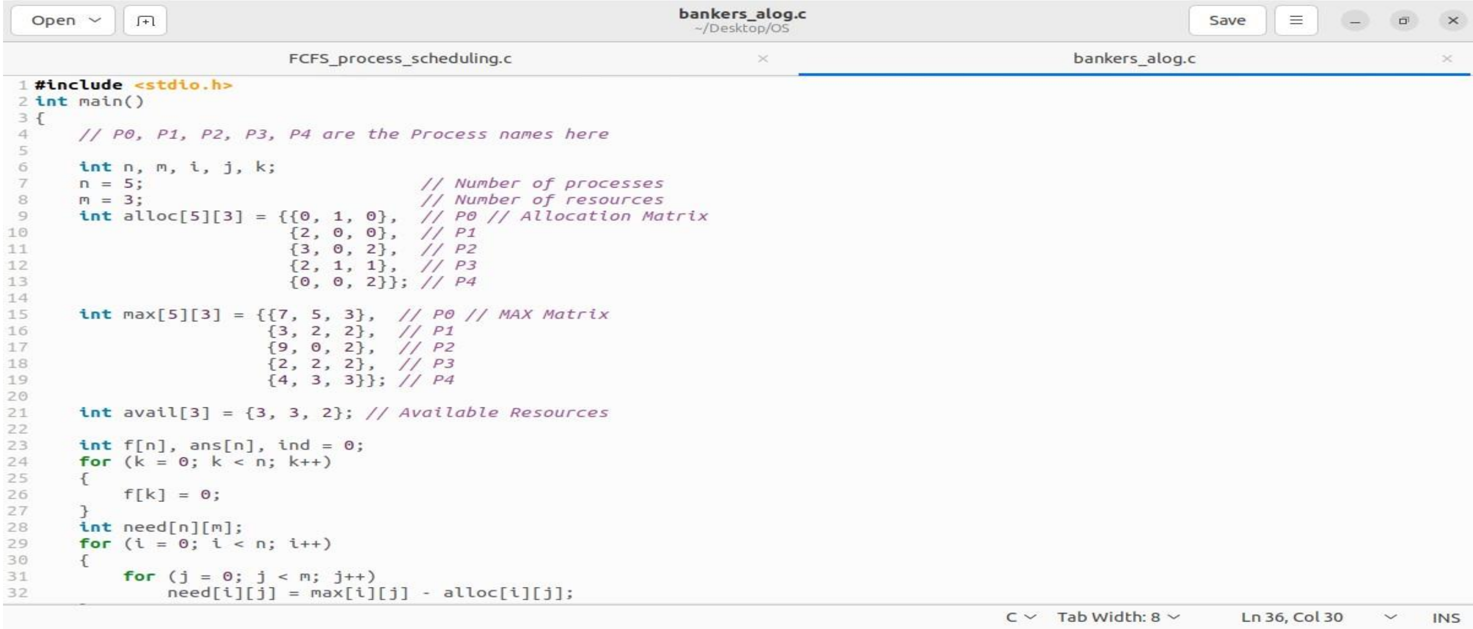
Prn: 12420020

Roll no: 2

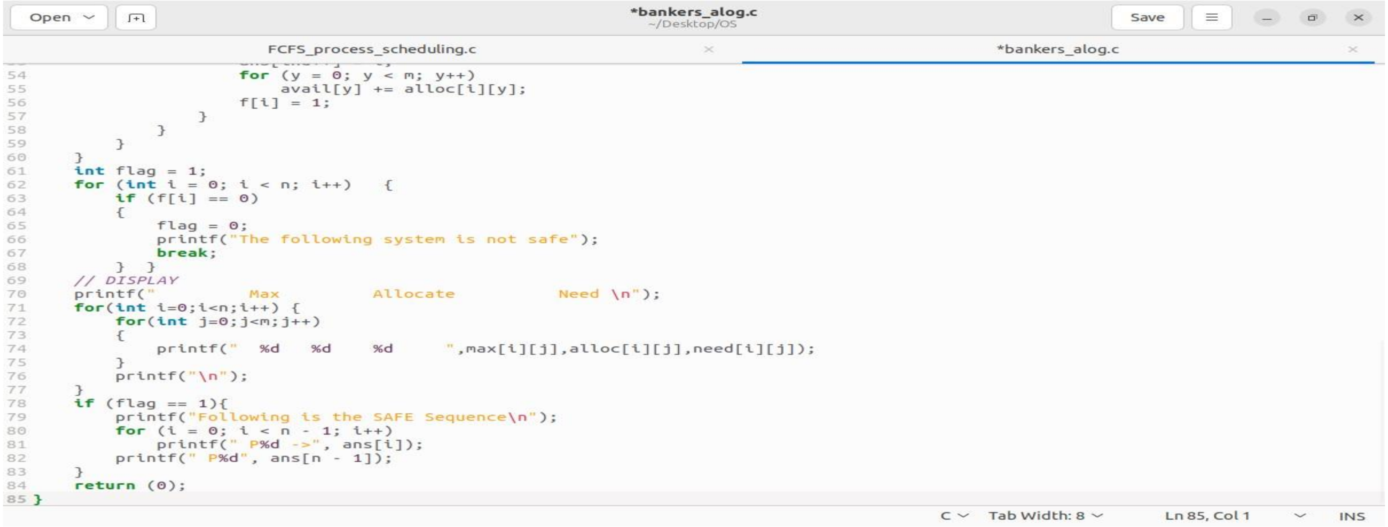
Sub: Operating System Lab

#Deadlock Handling Approach Algorithms. => avoids Deadlock into processes.

1. Q] Implement Banker's Algorithm.

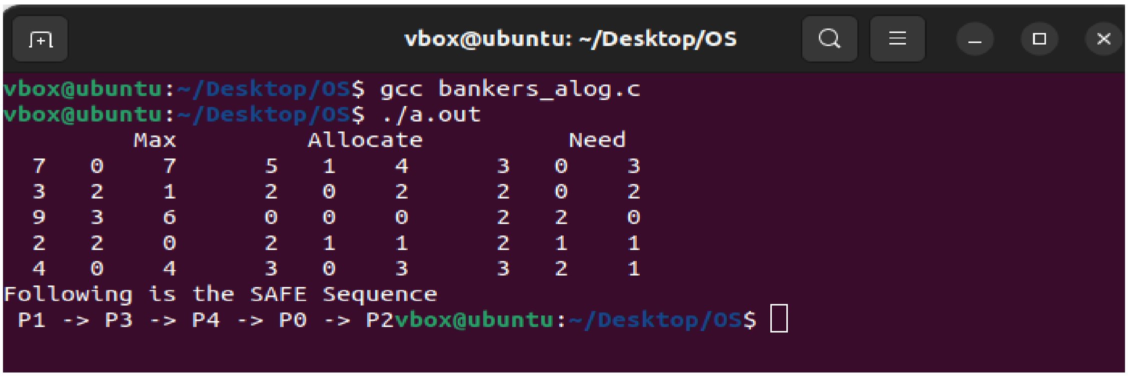


```c
#include <stdio.h>
int main()
{
    // P0, P1, P2, P3, P4 are the Process names here

    int n, m, i, j, k;
    n = 5;                          // Number of processes
    m = 3;                          // Number of resources
    int alloc[5][3] = {{0, 1, 0},  // P0 // Allocation Matrix
                       {2, 0, 0},  // P1
                       {3, 0, 2},  // P2
                       {2, 1, 1},  // P3
                       {0, 0, 2}}; // P4

    int max[5][3] = {{7, 5, 3},  // P0 // MAX Matrix
                     {3, 2, 2},  // P1
                     {9, 0, 2},  // P2
                     {2, 2, 2},  // P3
                     {4, 3, 3}}; // P4

    int avail[3] = {3, 3, 2}; // Available Resources

    int f[n], ans[n], ind = 0;
    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];
```



```c
                for (y = 0; y < m; y++)
                    avail[y] += alloc[i][y];
                f[i] = 1;
            }
        }
    }
}
int flag = 1;
for (int i = 0; i < n; i++)    {
    if (f[i] == 0)
    {
        flag = 0;
        printf("The following system is not safe");
        break;
    }  }
// DISPLAY
printf("             Max           Allocate              Need \n");
for(int i=0;i<n;i++) {
    for(int j=0;j<m;j++)
    {
        printf("  %d    %d     %d       ",max[i][j],alloc[i][j],need[i][j]);
    }
    printf("\n");
}
if (flag == 1){
    printf("Following is the SAFE Sequence\n");
    for (i = 0; i < n - 1; i++)
        printf(" P%d ->", ans[i]);
    printf(" P%d", ans[n - 1]);
}
return (0);
}
```

Output ===

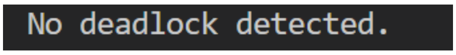


```
vbox@ubuntu:~/Desktop/OS$ gcc bankers_alog.c
vbox@ubuntu:~/Desktop/OS$ ./a.out
             Max           Allocate              Need
  7    0     7       5    1     4       3    0     3
  3    2     1       2    0     2       2    0     2
  9    3     6       0    0     0       2    2     0
  2    2     0       2    1     1       2    1     1
  4    0     4       3    0     3       3    2     1
Following is the SAFE Sequence
 P1 -> P3 -> P4 -> P0 -> P2vbox@ubuntu:~/Desktop/OS$ 
```

Q] Deadlock Detection Algorithm

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

#define P 5  // Number of processes
#define R 3  // Number of resources

int allocation[P][R]; // Allocation matrix
int max[P][R];        // Maximum matrix
int need[P][R];       // Need matrix
int available[R];     // Available resources

// Function to calculate the need matrix
void calculateNeed() {
    for (int i = 0; i < P; i++) {
        for (int j = 0; j < R; j++) {
            need[i][j] = max[i][j] - allocation[i][j];
        }
    }
}
// Function to detect deadlock using the wait-for graph
bool isDeadlocked() {
    bool finish[P] = {false};
    int count = 0;

    while (count < P) {
        bool found = false;

        for (int p = 0; p < P; p++) {
            // Check if process p can finish
            if (!finish[p]) {
                bool canFinish = true;
                for (int j = 0; j < R; j++) {
                    if (need[p][j] > available[j]) {
                        canFinish = false;
                        break;
                    }
                }
                if (canFinish) {
                    // Simulate finishing process p
                    for (int j = 0; j < R; j++) {
                        available[j] += allocation[p][j]; // Release allocated resources
                    }
                    finish[p] = true;
                    count++;
                    found = true;
                }
            }
        }
        if (!found) {
            // If no process could finish, we have a deadlock
            printf("Deadlock detected among the following processes:\n");
            for (int i = 0; i < P; i++) {
                if (!finish[i]) {
                    printf("Process %d\n", i);
                }
            }
            return true;
        }
    }

    printf("No deadlock detected.\n");
    return false;
}

int main() {
    // Initialize allocation, maximum, and available resources
    int allocationInput[P][R] = {
        {0, 1, 0},
        {2, 0, 0},
        {3, 0, 2},
        {2, 1, 1},
        {0, 0, 2}
    };

    int maxInput[P][R] = {
        {7, 5, 3},
        {3, 2, 2},
        {9, 0, 2},
        {2, 2, 2},
        {4, 3, 3}
    };

    int availableInput[R] = {3, 3, 2};

    // Copy inputs to global matrices
    for (int i = 0; i < P; i++) {
        for (int j = 0; j < R; j++) {
            allocation[i][j] = allocationInput[i][j];
            max[i][j] = maxInput[i][j];
        }
    }

    for (int i = 0; i < R; i++) {
        available[i] = availableInput[i];
    }

    // Calculate the need matrix
    calculateNeed();

    // Check for deadlock
    isDeadlocked();

    return 0;
}
```

Output ===



```
No deadlock detected.
```