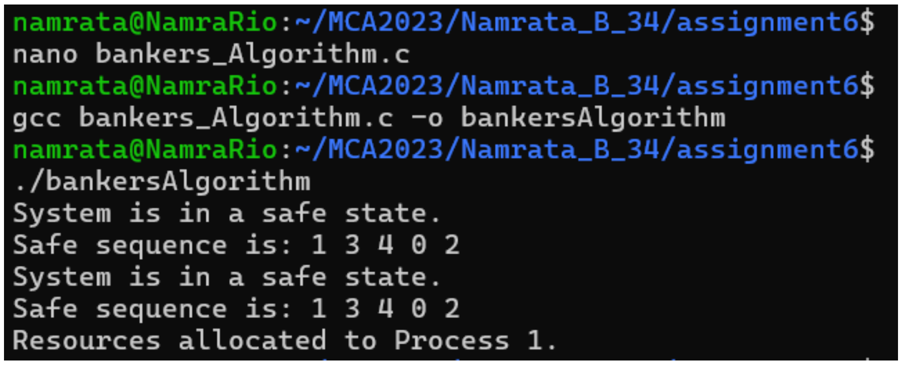
Assignment - 6

Topic: Deadlock Avoidance (Banker’s Algorithm)

Q: For deadlock avoidance, write a C program to simulate the Bankers algorithm.

CODE :

```
#include <stdio.h>
#define P 5
#define R 3
int isSafe(int processes[], int avail[], int max[][R], int allot[][R]) {
   int work[R];
   int finish[P] = {0};
   int safeSeq[P];
   int count = 0;
   for (int i = 0; i < R; i++) {
      work[i] = avail[i];
   }
   while (count < P) {
      int found = 0;
      for (int p = 0; p < P; p++) {
         if (finish[p] == 0) {
            int j;
            for (j = 0; j < R; j++) {
               if (max[p][j] - allot[p][j] > work[j]) {
                  break;
               }
            }
            if (j == R) {
               for (int k = 0; k < R; k++) {
                  work[k] += allot[p][k];
               }
               safeSeq[count++] = p;
               finish[p] = 1;
               found = 1;
            }
         }
      }
      if (found == 0) {
         printf("System is not in a safe state\n");
         return 0;
      }
   }
   printf("System is in a safe state.\nSafe sequence is: ");
   for (int i = 0; i < P; i++) {
      printf("%d ", safeSeq[i]);
   }
   printf("\n");
   return 1;
}
int requestResources(int processes[], int avail[], int max[][R], int allot[][R], int req[], int pid) {
    for (int i = 0; i < R; i++) {
        if (req[i] > max[pid][i] - allot[pid][i]) {
            printf("Error: Process %d has exceeded maximum claim.\n", pid);
            return 0;
        }
    }
    for (int i = 0; i < R; i++) {
        if (req[i] > avail[i]) {
            printf("Process %d must wait, resources not available.\n", pid);
            return 0;
        }
    }
    for (int i = 0; i < R; i++) {
        avail[i] -= req[i];
        allot[pid][i] += req[i];
    }
    if (isSafe(processes, avail, max, allot)) {
        printf("Resources allocated to Process %d.\n", pid);
        return 1;
    }
    for (int i = 0; i < R; i++) {
        avail[i] += req[i];
        allot[pid][i] -= req[i];
    }
    printf("Process %d must wait, as system would become unsafe.\n", pid);
    return 0;
}
int main() {
    int processes[P] = {0, 1, 2, 3,
4};
    int avail[R] = {3, 3, 2};
    int max[P][R] = {
        {7, 5, 3},
        {3, 2, 2},
        {9, 0, 2},
        {2, 2, 2},
        {4, 3, 3}
    };
    int allot[P][R] = {
        {0, 1, 0},
        {2, 0, 0},
        {3, 0, 2},
        {2, 1, 1},
        {0, 0, 2}
    };
    isSafe(processes, avail, max, allot);
    int req1[R] = {1, 0, 2};
    requestResources(processes, avail, max, allot, req1, 1);
    return 0;
}
```



```
namrata@NamraRio:~/MCA2023/Namrata_B_34/assignment6$
nano bankers_Algorithm.c
namrata@NamraRio:~/MCA2023/Namrata_B_34/assignment6$
gcc bankers_Algorithm.c -o bankersAlgorithm
namrata@NamraRio:~/MCA2023/Namrata_B_34/assignment6$
./bankersAlgorithm
System is in a safe state.
Safe sequence is: 1 3 4 0 2
System is in a safe state.
Safe sequence is: 1 3 4 0 2
Resources allocated to Process 1.
```