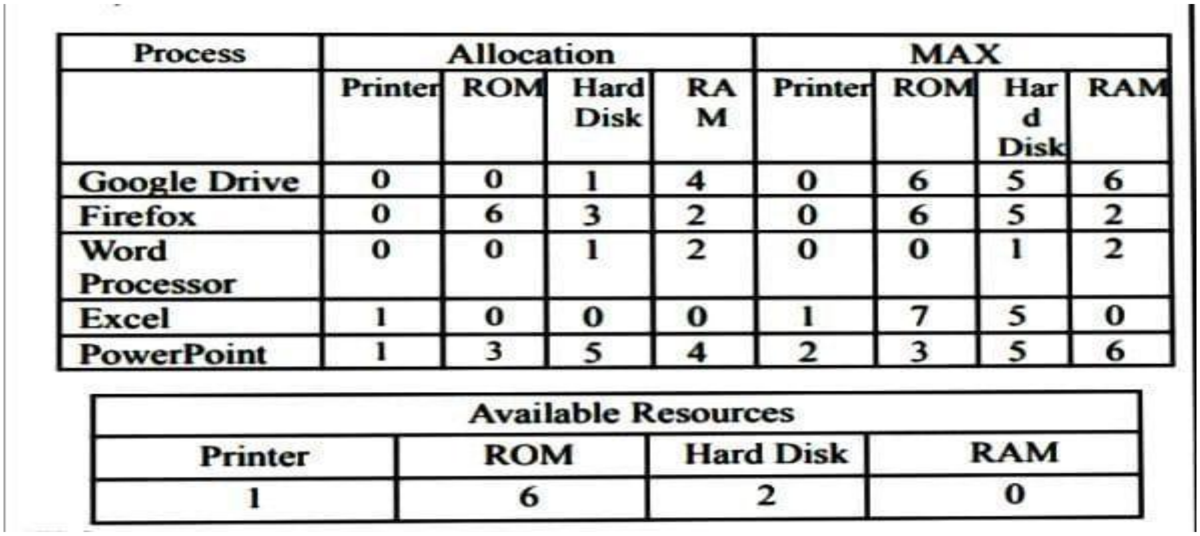
7. [Deadlock Handling]

A computer system has four different types of resources (printer, ROM, hard disk and RAM) and it needs to complete execution of five processes (Google Drive, Firefox, Word Processor, Excel and PowerPoint). The number of allocated resources, maximum needed resources to complete the execution and total available resources should be taken from the user. (Sample data for reference is given below)



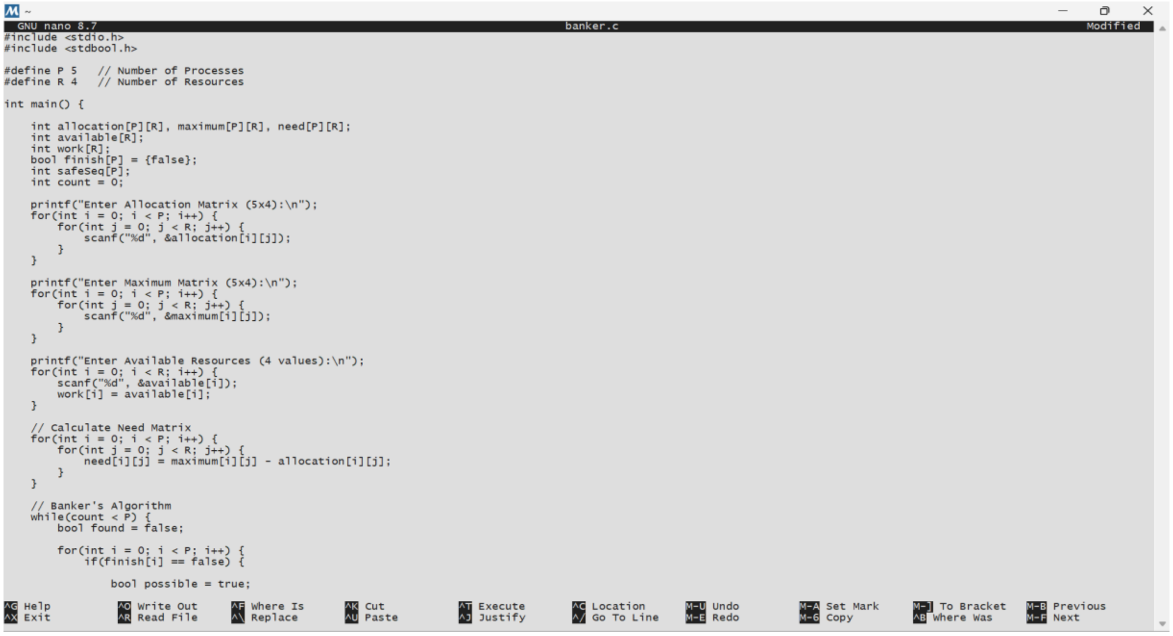
Process	Allocation				MAX			
	Printer	ROM	Hard Disk	RAM	Printer	ROM	Hard Disk	RAM
Google Drive	0	0	1	4	0	6	5	6
Firefox	0	6	3	2	0	6	5	2
Word Processor	0	0	1	2	0	0	1	2
Excel	1	0	0	0	1	7	5	0
PowerPoint	1	3	5	4	2	3	5	6

Available Resources			
Printer	ROM	Hard Disk	RAM
1	6	2	0

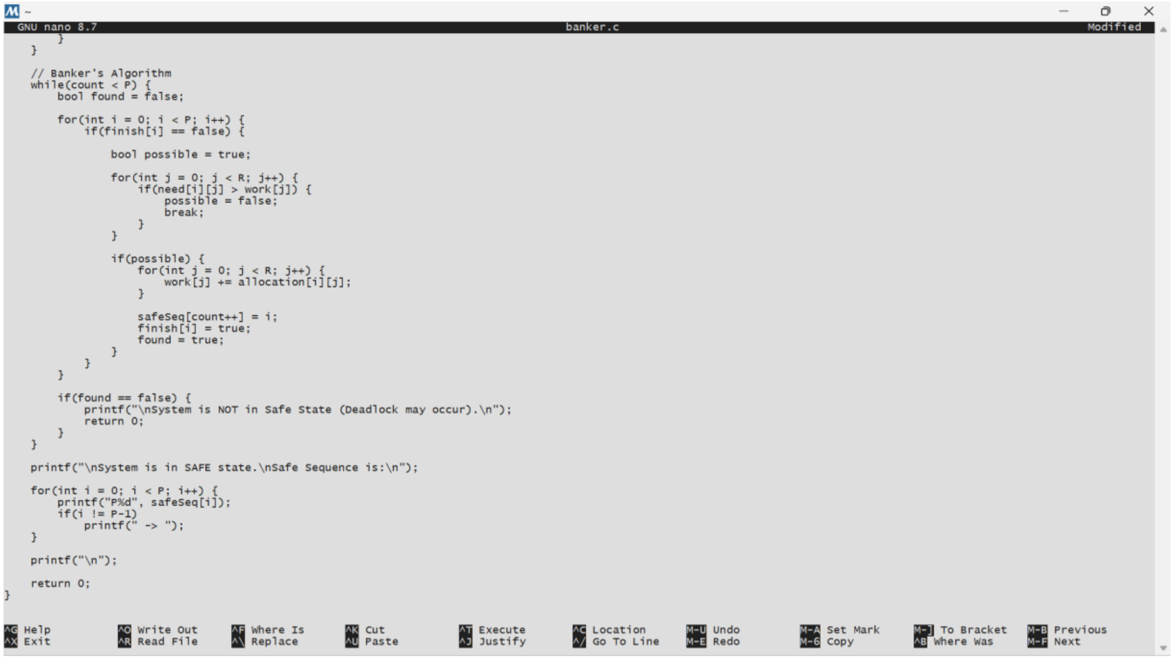
Write a program to:

- Calculate current need of resources by each process.
- Find whether these processes can be executed with available resources. If yes, show the correct order of process execution.

CODE :

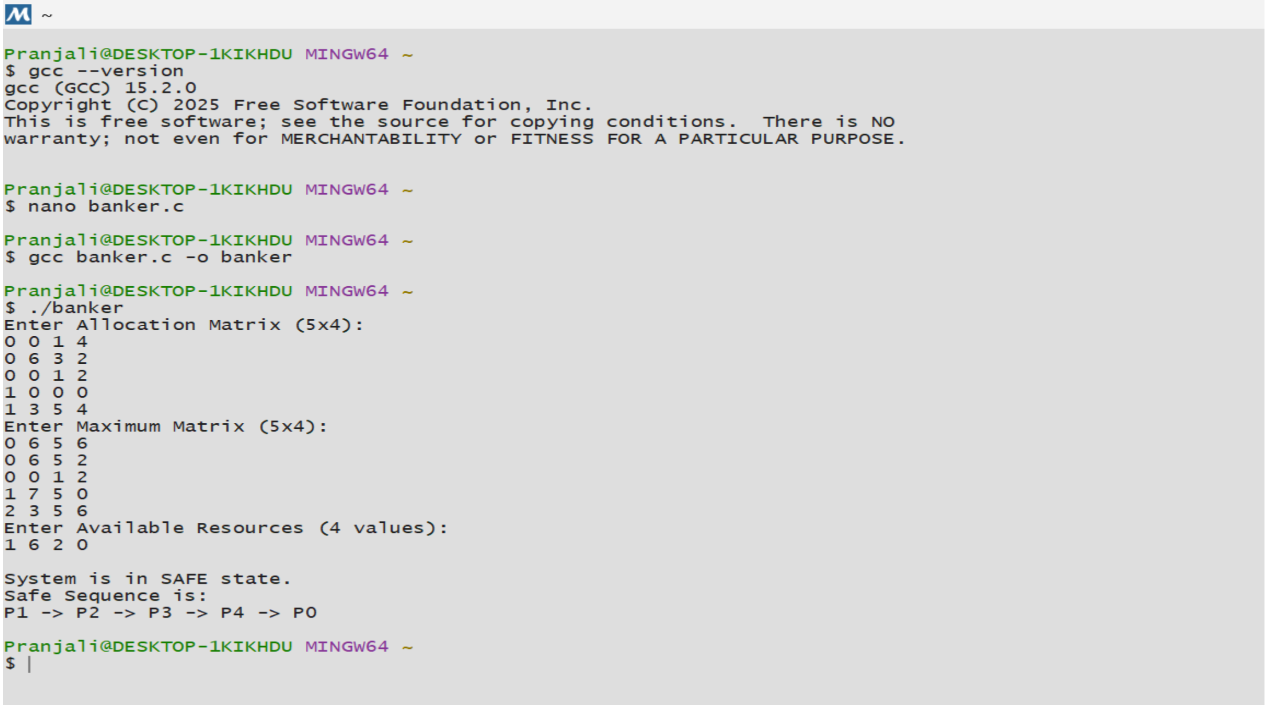


```
#include <stdio.h>
#include <stdbool.h>

#define P 5   // Number of Processes
#define R 4   // Number of Resources

int main() {

    int allocation[P][R], maximum[P][R], need[P][R];
    int available[R];
    int work[R];
    bool finish[P] = {false};
    int safeSeq[P];
    int count = 0;

    printf("Enter Allocation Matrix (5x4):\n");
    for(int i = 0; i < P; i++) {
        for(int j = 0; j < R; j++) {
            scanf("%d", &allocation[i][j]);
        }
    }

    printf("Enter Maximum Matrix (5x4):\n");
    for(int i = 0; i < P; i++) {
        for(int j = 0; j < R; j++) {
            scanf("%d", &maximum[i][j]);
        }
    }

    printf("Enter Available Resources (4 values):\n");
    for(int i = 0; i < R; i++) {
        scanf("%d", &available[i]);
        work[i] = available[i];
    }

    // Calculate Need Matrix
    for(int i = 0; i < P; i++) {
        for(int j = 0; j < R; j++) {
            need[i][j] = maximum[i][j] - allocation[i][j];
        }
    }

    // Banker's Algorithm
    while(count < P) {
        bool found = false;

        for(int i = 0; i < P; i++) {
            if(finish[i] == false) {

                bool possible = true;
```



```
        }
    }

    // Banker's Algorithm
    while(count < P) {
        bool found = false;

        for(int i = 0; i < P; i++) {
            if(finish[i] == false) {

                bool possible = true;

                for(int j = 0; j < R; j++) {
                    if(need[i][j] > work[j]) {
                        possible = false;
                        break;
                    }
                }

                if(possible) {
                    for(int j = 0; j < R; j++) {
                        work[j] += allocation[i][j];
                    }

                    safeSeq[count++] = i;
                    finish[i] = true;
                    found = true;
                }
            }
        }

        if(found == false) {
            printf("\nSystem is NOT in Safe State (Deadlock may occur).\n");
            return 0;
        }
    }

    printf("\nSystem is in SAFE state.\nSafe Sequence is:\n");

    for(int i = 0; i < P; i++) {
        printf("P%d", safeSeq[i]);
        if(i != P-1)
            printf(" -> ");
    }

    printf("\n");

    return 0;
}
```

OUTPUT :



```
Pranjali@DESKTOP-1KIKHDU MINGW64 ~
$ gcc --version
gcc (GCC) 15.2.0
Copyright (C) 2025 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.


Pranjali@DESKTOP-1KIKHDU MINGW64 ~
$ nano banker.c

Pranjali@DESKTOP-1KIKHDU MINGW64 ~
$ gcc banker.c -o banker

Pranjali@DESKTOP-1KIKHDU MINGW64 ~
$ ./banker
Enter Allocation Matrix (5x4):
0 0 1 4
0 6 3 2
0 0 1 2
1 0 0 0
1 3 5 4
Enter Maximum Matrix (5x4):
0 6 5 6
0 6 5 2
0 0 1 2
1 7 5 0
2 3 5 6
Enter Available Resources (4 values):
1 6 2 0

System is in SAFE state.
Safe Sequence is:
P1 -> P2 -> P3 -> P4 -> P0

Pranjali@DESKTOP-1KIKHDU MINGW64 ~
$ |
```