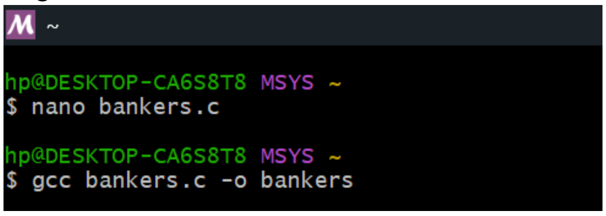
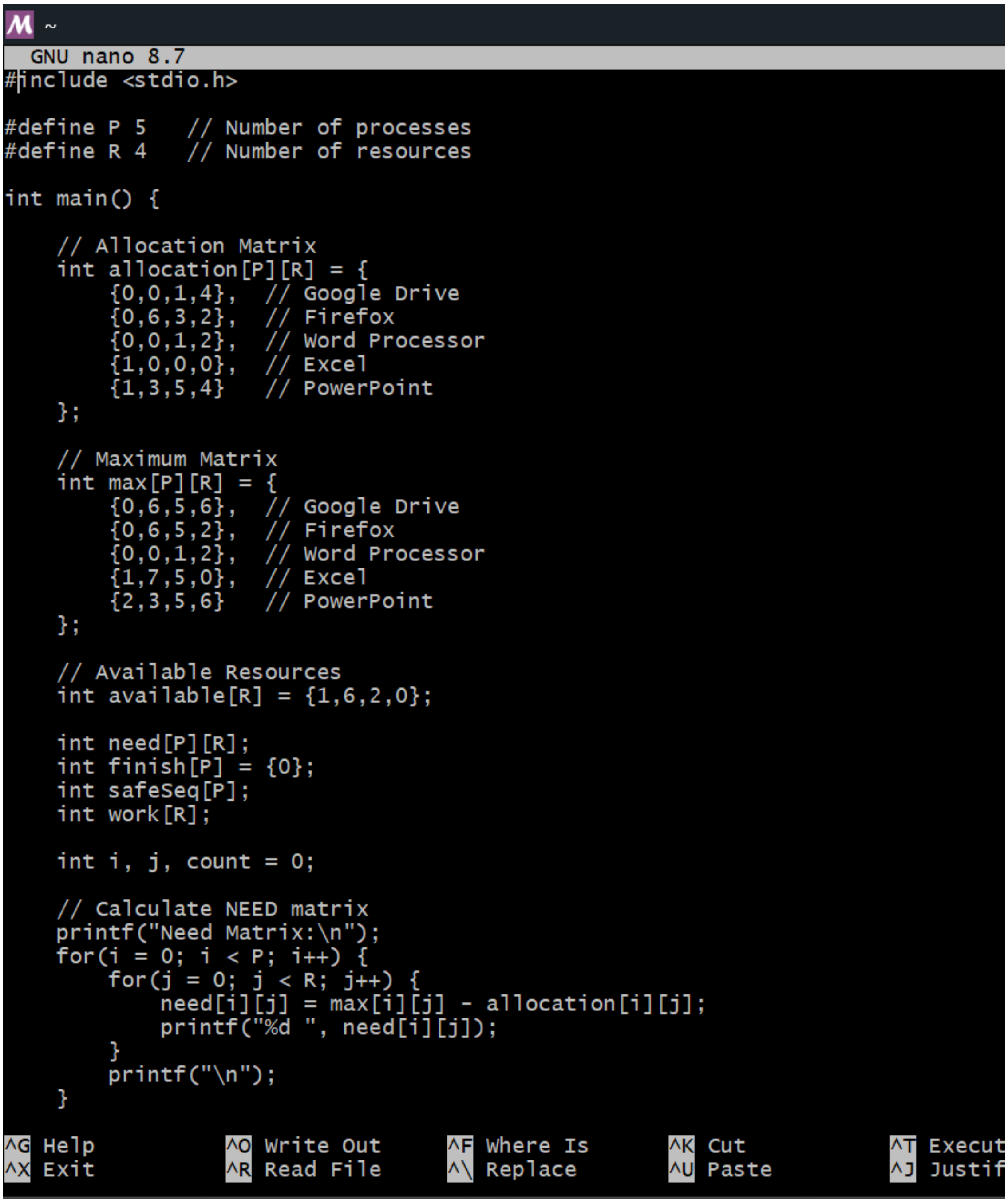
Practical 7:

Aim: Develop a program to manage resource allocation for five processes (Google Drive, Firefox, Word Processor, Excel, and PowerPoint) using four types of resources (Printer, ROM, Hard Disk, and RAM). The program takes input for allocated, maximum, and available resources, calculates the current need of each process, and determines if a safe execution order exists using the Banker's Algorithm.

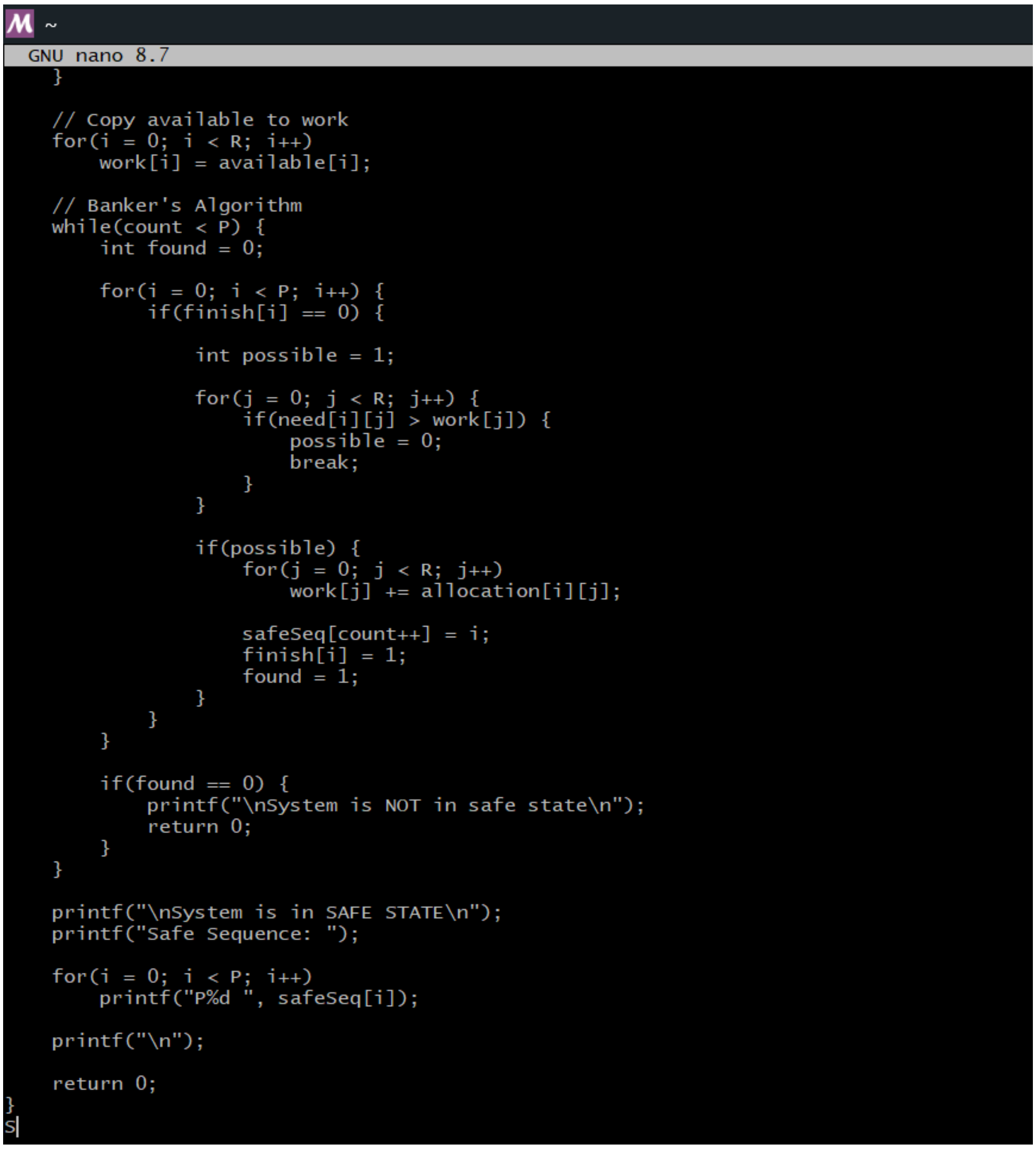


```
~

hp@DESKTOP-CA6S8T8 MSYS ~
$ nano bankers.c

hp@DESKTOP-CA6S8T8 MSYS ~
$ gcc bankers.c -o bankers
```

Code:

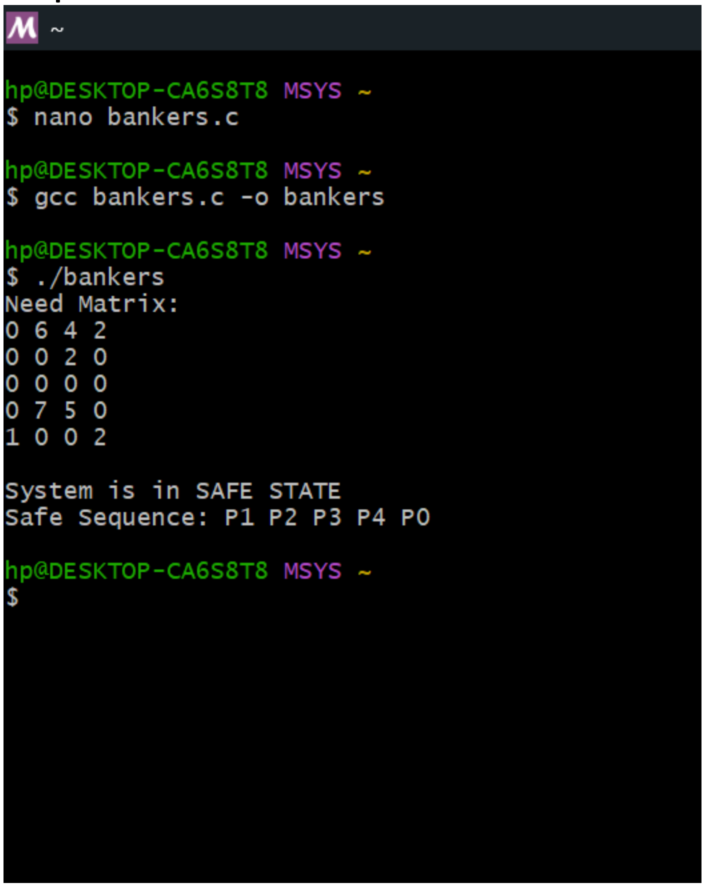


```
~
  GNU nano 8.7
#include <stdio.h>

#define P 5   // Number of processes
#define R 4   // Number of resources

int main() {

    // Allocation Matrix
    int allocation[P][R] = {
        {0,0,1,4},   // Google Drive
        {0,6,3,2},   // Firefox
        {0,0,1,2},   // Word Processor
        {1,0,0,0},   // Excel
        {1,3,5,4}    // PowerPoint
    };

    // Maximum Matrix
    int max[P][R] = {
        {0,6,5,6},   // Google Drive
        {0,6,5,2},   // Firefox
        {0,0,1,2},   // Word Processor
        {1,7,5,0},   // Excel
        {2,3,5,6}    // PowerPoint
    };

    // Available Resources
    int available[R] = {1,6,2,0};

    int need[P][R];
    int finish[P] = {0};
    int safeSeq[P];
    int work[R];

    int i, j, count = 0;

    // Calculate NEED matrix
    printf("Need Matrix:\n");
    for(i = 0; i < P; i++) {
        for(j = 0; j < R; j++) {
            need[i][j] = max[i][j] - allocation[i][j];
            printf("%d ", need[i][j]);
        }
        printf("\n");
    }

^G Help          ^O Write Out     ^F Where Is      ^K Cut           ^T Execut
^X Exit          ^R Read File     ^\ Replace       ^U Paste         ^J Justif
```



```
M ~
  GNU nano 8.7
    }

    // Copy available to work
    for(i = 0; i < R; i++)
        work[i] = available[i];

    // Banker's Algorithm
    while(count < P) {
        int found = 0;

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

                int possible = 1;

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

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

                    safeSeq[count++] = i;
                    finish[i] = 1;
                    found = 1;
                }
            }
        }

        if(found == 0) {
            printf("\nSystem is NOT in safe state\n");
            return 0;
        }
    }

    printf("\nSystem is in SAFE STATE\n");
    printf("Safe Sequence: ");

    for(i = 0; i < P; i++)
        printf("P%d ", safeSeq[i]);

    printf("\n");

    return 0;
}
S
```

Output :



```
~

hp@DESKTOP-CA6S8T8 MSYS ~
$ nano bankers.c

hp@DESKTOP-CA6S8T8 MSYS ~
$ gcc bankers.c -o bankers

hp@DESKTOP-CA6S8T8 MSYS ~
$ ./bankers
Need Matrix:
0 6 4 2
0 0 2 0
0 0 0 0
0 7 5 0
1 0 0 2

System is in SAFE STATE
Safe Sequence: P1 P2 P3 P4 P0

hp@DESKTOP-CA6S8T8 MSYS ~
$
```