```c
#include <stdio.h>
#include <stdlib.h>
#define MAX 100

int main()
{
    FILE *fp;
    char fileName[MAX], line [MAX];
    int P = 0, R = 0, i, j, z, element;
    char str [10];
    char *pch;
     int K[P], flag = 1, count = 0, safe[P], out [P], o = 0;

    printf("Enter the file name:\t");
    scanf("%s",fileName);

    fp = fopen(fileName,"r");
    while ( fp == NULL )
    {
         system("cls");
         printf("File ( %s ) dose not exist!\n", fileName);
         printf("Enter the file name:\t");
            scanf("%s",fileName);
            fp = fopen(fileName,"r");
     }
    //Reading the first two lines P & R
    fgets(line, sizeof line, fp);
    removeNewLine(line);
    P = atoi (line);

    fgets(line, sizeof line, fp);
    removeNewLine(line);
    R = atoi (line);

    //MAXIMUM array to hold the maximum needs of resources the set of processes P
    int MAXIMUM[P][R];

    //ALLOCATION array to hold the currently allocated resources of the set of processes P
    int ALLOCATION[P][R];

    //AVAILABLE array to hold the currently available resources
    int AVAILABLE[R];

    //NEEDS array to hold the currently needed resources of the set of processes P
    int NEEDS[P][R];


    fgets(line, sizeof line, fp);//to remove the line
    //loop to read the MAXIMUM array
    for (i = 0; i < P; i++)
    {
        j = 0;
        fgets(line, sizeof line, fp);
        removeNewLine(line);

        pch = strtok (line," ");//Split by space

        while(pch != NULL)
        {
            strcpy(str, pch);
            element = atoi (str);
            MAXIMUM[i][j] = element;
            j++;
            pch = strtok (NULL, " ");
        }

    }

    fgets(line, sizeof line, fp);//to remove the line
    //loop to read the ALLOCATION array
    for (i = 0; i < P; i++)
    {
        j = 0;
        fgets(line, sizeof line, fp);
        removeNewLine(line);

        pch = strtok (line," ");//Split by space

        while(pch != NULL)
        {
            strcpy(str, pch);
            element = atoi (str);
            ALLOCATION[i][j] = element;
            j++;
```

```
85                  pch = strtok (NULL, " ");
86              }
87
88          }
89
90          //reading the AVILABLE array
91          fgets(line, sizeof line, fp);
92          j = 0;
93          fgets(line, sizeof line, fp);
94          removeNewLine(line);
95
96          pch = strtok (line," ");//Split by space
97
98          while(pch != NULL)
99          {
100             strcpy(str, pch);
101             element = atoi (str);
102             AVAILABLE[j] = element;
103             j++;
104             pch = strtok (NULL, " ");
105             }
106
107         fclose(fp);
108
109         //calculate the NEEDS array
110         for ( i = 0; i < P; i++)
111         {
112             for ( j = 0; j < R; j ++)
113             {
114                 NEEDS[i][j] = MAXIMUM[i][j] - ALLOCATION [i][j];
115             }
116
117         }
118
119
120
121         printf("P = %d \t R = %d\n", P, R);
122         printf("\nThe MAXIMUM array:\n");
123         for ( i = 0; i < P; i++)
124         {
125             for ( j = 0; j < R; j ++)
126             {
127                 printf("%d\t", MAXIMUM[i][j]);
128             }
129             printf("\n");
130         }
131         printf("\nThe ALLOCATION array:\n");
132         for ( i = 0; i < P; i++)
133         {
134             for ( j = 0; j < R; j ++)
135             {
136                 printf("%d\t", ALLOCATION[i][j]);
137             }
138             printf("\n");
139         }
140
141          printf("\nThe NEEDS array:\n");
142         for ( i = 0; i < P; i++)
143         {
144             for ( j = 0; j < R; j ++)
145             {
146                 printf("%d\t", NEEDS[i][j]);
147             }
148             printf("\n");
149         }
150
151         printf("\nThe AVAILABLE array:\n");
152         for ( i = 0; i < R; i++)
153         {
154             printf("%d\t", AVAILABLE[i]);
155         }
156         printf("\n");
157
158
159             //Check if the system is safe
160             //Banker algorithm
161
162          for(i = 0;i < P; i++)
163              {
164               K[i] = 1;
165              }
166
167     while(flag)  //flag for loop correct continuity
168     {
```

```
169             flag = 0;
170               for(i = 0; i < P; i++)
171             {
172                       int c = 0;
173                       for(j = 0; j < R; j++)
174                       {
175                             if((K[i]== 1)&&(NEEDS[i][j] <= AVAILABLE[j]))
176                             {
177                                 c++;
178                                 if(c == R)
179                                  {
180                                     for(z = 0; z < R; z++)
181                                     {
182                                             AVAILABLE[z] += ALLOCATION[i][j];
183                                             K[i] = 0;
184                                             flag = 1;
185                                     }
186                                     out[o] = i;
187                                     o++;
188                                     if(K[i] == 0)
189                                     {
190                                         i = P;
191                                     }
192                                  }
193                             }
194                       }
195                }
196     }
197
198     for(i = 0; i < P; i++)
199     {
200         if(K[i]== 0)
201         {
202           count++;
203         }
204         else
205         {
206           out[o] = i;
207           o++;
208         }
209     }
210
211     if(count == P)
212     {
213        printf("\n The system is in safe state");
214        printf("\n< ");
215        for (i = 0; i < P-1; i++)
216        {
217            printf("P%d->", out[i]);
218        }
219        printf("P%d >", out[P-1]);
220     }
221     else
222     {
223        printf("\n System are in dead lock");
224        printf("\n System is in unsafe state");
225     }
226        return 0;
227   }
228
229    //Function to remove \n from a string
230    void removeNewLine(char *str)
231   {
232     char *p1 = str, *p2 = str;
233     do
234       while (*p2 == '\n')
235         p2++;
236     while (*p1++ = *p2++);
237   }
238
```