Operating Systems

<u>Experiment No. 7</u>

Name: Ayush Jain

SAPID: 60004200132

Batch: B1

Branch: Computer Engineering

Aim- Implement order scheduling in supply chain using Banker's Algorithm

Problem Statement-

In supply chain, a deadlock can frequently occur when managing supply and demand between supplier and retailers. Supplier needs to reasonably arrange sequence of order to satisfy retailers. Scheduling management for orders is critical in product supply chain to ensure every retailer to receive commodities from a common supplier. The order contracts between supplier and retailers can specify various payment options such as before or after the delivery of commodities or even several installments. With such a variation in money flow, a supplier has to manage not only the cost for manufacturing commodities but also other operational cost such as management fee, equipment maintenance and storage charge. These situations may cause supplier to reach insufficient money flow to produce commodities, resulting in a low productivity or even a state of financial deficit. Hence, the risk of capital chain rupture for supplier increases with the increase in number of orders or amounts of commodities. Without a careful scheduling of orders, resources allocation of supplier may result in a deadlock situation. Therefore, efficient algorithm is required to avoid such deadlock situations leading to capital chain rupture in supply chain management.

In order to avoid capital chain rupture in OSM and to effectively manage the money flow, it is necessary to adequately allocate requirements of commodities to ensure satisfaction of retailers.

- A single order in OSM as a process and multiple commodities as resources that retailers need.
- If scheduling in OSM is inadequate, it will cause insufficient money flow of supplier and create a risk of capital chain rupture.
- Deadlock will arise when supplier cannot produce commodities for current order and retailers enter an unsafe state of indefinite waiting.
- Since original resources (commodities) do not return but payment is made from retailers to supplier in OSM, define a safe as a state with sufficient money flow for the next production.
- Use Banker’s algorithm to acquire a safe sequence in OSM where the money flow remains in a safe state

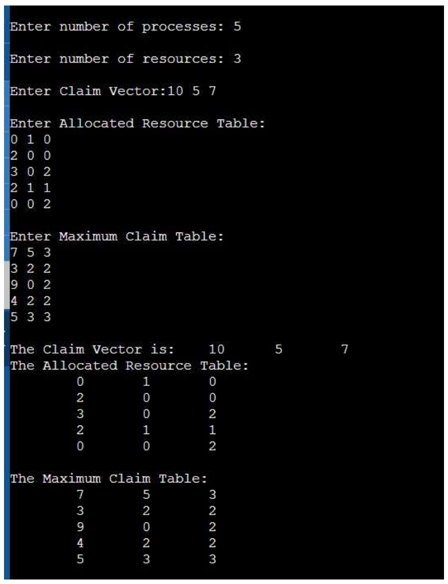
Code:

```
#include <stdio.h>

int current[5][5], maximum_claim[5][5], available[5]; int
allocation[5] = {0, 0, 0, 0, 0}; int maxres[5], running[5], safe =
0; int counter = 0, i, j, exec, resources, processes, k = 1;

int main()
{
printf("\nEnter number of processes: ");      scanf("%d",
&processes);

    for (i = 0; i < processes; i++)
{        running[i] = 1;
counter++;
    }

    printf("\nEnter number of resources: ");      scanf("%d",
&resources);

    printf("\nEnter Claim Vector:");
     for (i = 0; i < resources; i++)
```

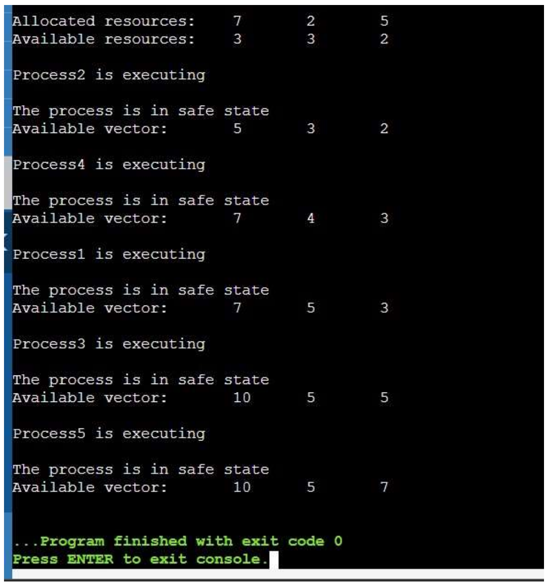
```
{
       scanf("%d", &maxres[i]);
     }

   printf("\nEnter Allocated Resource Table:\n");      for (i =
0; i < processes; i++)
{        for(j = 0; j < resources; j++)
{
   scanf("%d", &current[i][j]);
         }
     }

     printf("\nEnter Maximum Claim Table:\n");      for (i =
0; i < processes; i++)
{         for(j = 0; j < resources; j++)
{
             scanf("%d", &maximum_claim[i][j]);
         }
     }

printf("\nThe Claim Vector is: ");      for (i
= 0; i < resources; i++)
{        printf("\t%d", maxres[i]);
}

                 printf("\nThe Allocated Resource Table:\n");
     for (i = 0; i < processes; i++)
{        for (j = 0; j < resources; j++)
{             printf("\t%d", current[i][j]);
         } printf("\n");
     }

     printf("\nThe Maximum Claim Table:\n");      for (i =
0; i < processes; i++)
{         for (j = 0; j < resources; j++)
{
       printf("\t%d", maximum_claim[i][j]);
        }
printf("\n");
     }
     for (i = 0; i < processes; i++)
{          for (j = 0; j < resources; j++)
{              allocation[j] += current[i][j];
         }
     }

     printf("\nAllocated resources:");
      for (i = 0; i < resources; i++)
{          printf("\t%d", allocation[i]);
     }

     for (i = 0; i < resources; i++)
{         available[i] = maxres[i] - allocation[i];
}

     printf("\nAvailable resources:");      for (i
= 0; i < resources; i++)
{          printf("\t%d", available[i]);
     }
printf("\n");

     while (counter != 0)
{          safe = 0;          for (i = 0; i <
processes; i++)
{              if (running[i])
{                  exec = 1;                  for (j = 0; j <
resources; j++)
{                      if (maximum_claim[i][j] - current[i][j] > available[j]) {
                         exec = 0;
break;
                     }
}                  if (exec)
{                      printf("\nProcess%d is executing\n", i + 1);
running[i] = 0;                      counter--;                      safe = 1;
                       for (j = 0; j < resources; j++)
{                          available[j] += current[i][j];
                     }
break;
```

```
                }
            }          }
if (!safe)
{
            printf("\nThe processes are in unsafe state.\n");              break;
        }
else
{                    printf("\nThe  process  is  in  safe  state");
printf("\nAvailable vector:");

            for (i = 0; i < resources; i++)
{                  printf("\t%d", available[i]);
            }
printf("\n");
        }
    }     return
0;
}
```

Output:



```
Enter number of processes: 5

Enter number of resources: 3

Enter Claim Vector:10 5 7

Enter Allocated Resource Table:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2

Enter Maximum Claim Table:
7 5 3
3 2 2
9 0 2
4 2 2
5 3 3

The Claim Vector is:    10      5       7
The Allocated Resource Table:
        0       1       0
        2       0       0
        3       0       2
        2       1       1
        0       0       2

The Maximum Claim Table:
        7       5       3
        3       2       2
        9       0       2
        4       2       2
        5       3       3
```



```
Allocated resources:    7       2       5
Available resources:    3       3       2

Process2 is executing

The process is in safe state
Available vector:       5       3       2

Process4 is executing

The process is in safe state
Available vector:       7       4       3

Process1 is executing

The process is in safe state
Available vector:       7       5       3

Process3 is executing

The process is in safe state
Available vector:       10      5       5

Process5 is executing

The process is in safe state
Available vector:       10      5       7


...Program finished with exit code 0
Press ENTER to exit console.
```