Experiment 4:

Subject: CSL403 Operating System Lab

NAME : GINI CHACKO

ROLL : 8942

CLASS : SE COMPS B

Aim: To study Deadlock detection and Avoidance strategies

Objectives: To gain practical experience with designing and implementing concepts of operating systems such as system calls, CPU scheduling, process management, memory management, file systems and deadlock handling using C language in Linux environment.

Problem Statement:
WAP for the following.
Inputs: Number of processes, No of Resources, Instances of each resources, Number of resources held by each process , Number of resources needed by each process/Maximum number of resources needed by each process.

Write a menu driven program.
1) Detect if a deadlock exists. Also show the processes involved in deadlock
2) Check if the deadlock can be avoided (using bankers algo.). If yes, give the safe state sequence.

Which data structure is used by OS to find deadlock in the system
Answer: In this approach, The OS doesn't apply any mechanism to avoid or prevent the deadlocks. Therefore the system considers that the deadlock will definitely occur. In order to get rid of deadlocks, The OS periodically checks the system for any deadlock. In case, it finds any of the deadlock then the OS will recover the system using some recovery techniques. The main task of the OS is detecting the deadlocks. The OS can detect the deadlocks with the help of Resource allocation graph. In single instanced resource types, if a cycle is being formed in the system then there will definitely be a deadlock. On the other hand, in multiple instanced resource type graph, detecting a cycle is not just enough. We have to apply the safety algorithm on the system by converting the resource allocation graph into the allocation matrix and request matrix. In order to recover the system from deadlocks, either OS considers resources or processes. ➢ **For Resource** **Preempt the resource :** We can snatch one of the resources from the owner of the resource (process) and give it to the other process with the expectation that it will complete the

execution and will release this resource sooner. Well, choosing a resource which will be snatched is going to be a bit difficult.

Rollback to a safe state : System passes through various states to get into the deadlock state. The operating system canrollback the system to the previous safe state. For this purpose, OS needs to implement check pointing at every state.

The moment, we get into deadlock, we will rollback all the allocations to get into the previous safe state.

- **For Process**

Kill a process : Killing a process can solve our problem but the bigger concern is to decide which process to kill. Generally, Operating system kills a process which has done least amount of work until now.

Kill all process : This is not a suggestible approach but can be implemented if the problem becomes very serious. Killing all process will lead to inefficiency in the system because all the processes will execute again from starting.

Program Section:

A.] DEADLOCK DETECTION

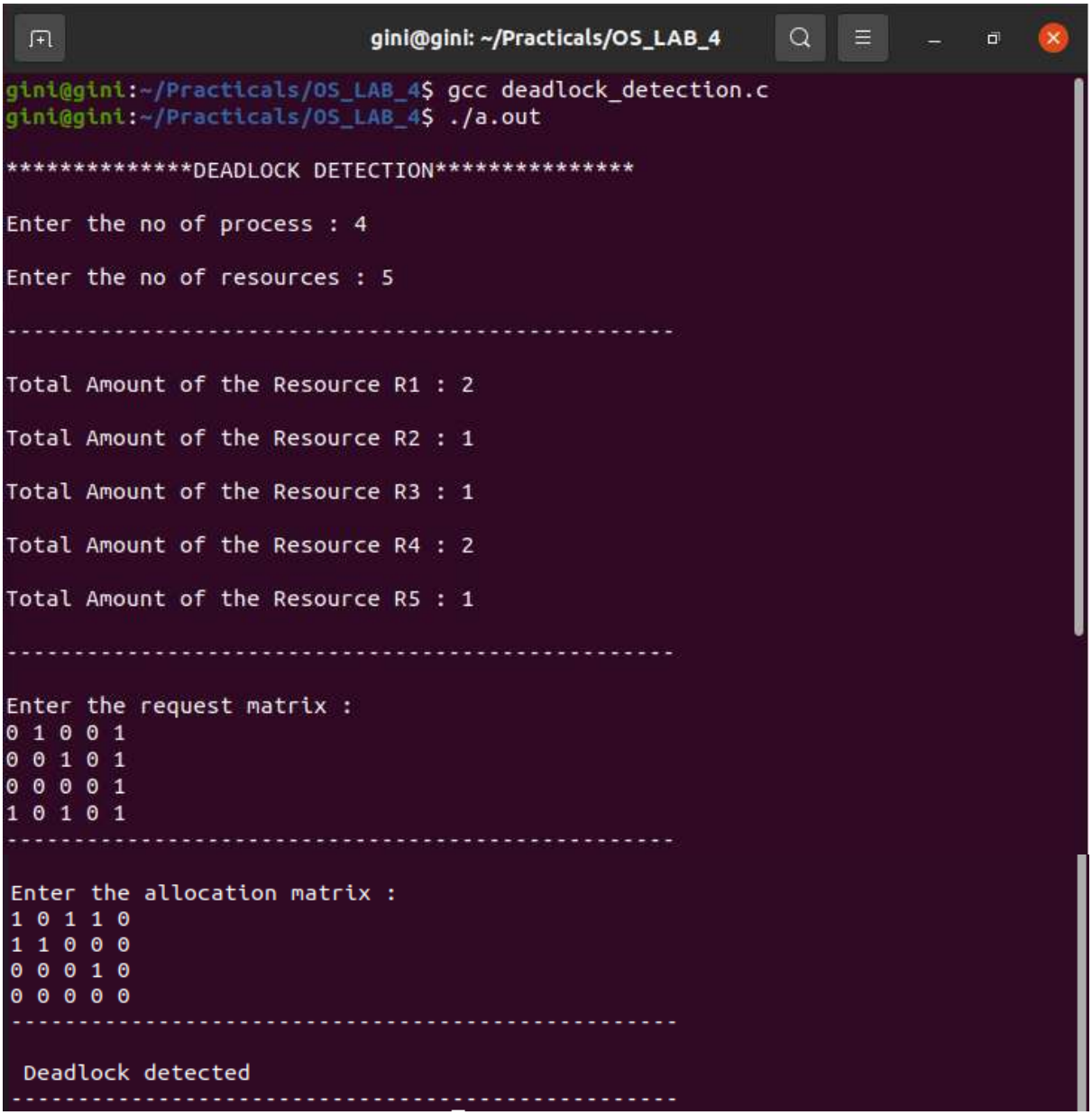
CODE:

```
#include<stdio.h>
static int mark[20];
int i, j, np, nr;

int main() {
  int alloc[10][10], request[10][10], avail[10], r[10], w[10];
  printf("\n***************DEADLOCK DETECTION****************\n");
  printf("\nEnter the no of process : ");
  scanf("%d", & np);
  printf("\nEnter the no of resources : ");
  scanf("%d", & nr);
  printf("\n-------------------------------------------------\n");
  for (i = 0; i < nr; i++) {
    printf("\nTotal Amount of the Resource R%d : ", i + 1);
    scanf("%d", & r[i]);
  }
  printf("\n-------------------------------------------------\n");
  printf("\nEnter the request matrix : \n");

  for (i = 0; i < np; i++)
    for (j = 0; j < nr; j++)
      scanf("%d", & request[i][j]);
  printf("-------------------------------------------------\n");
```

```
printf("\nEnter the allocation matrix : \n");
for (i = 0; i < np; i++)
  for (j = 0; j < nr; j++)
    scanf("%d", & alloc[i][j]);
printf("--------------------------------------------------\n");

/*Available Resource calculation*/
for (j = 0; j < nr; j++) {
  avail[j] = r[j];
  for (i = 0; i < np; i++) {
    avail[j] -= alloc[i][j];
  }
}

//marking processes with zero allocation
for (i = 0; i < np; i++) {
  int count = 0;
  for (j = 0; j < nr; j++) {
    if (alloc[i][j] == 0)
      count++;
    else
      break;
  }
  if (count == nr)
    mark[i] = 1;
}

// initialize W with avail

for (j = 0; j < nr; j++)
  w[j] = avail[j];

//mark processes with request less than or equal to W
for (i = 0; i < np; i++) {
  int canbeprocessed = 0;
  if (mark[i] != 1) {
    for (j = 0; j < nr; j++) {
      if (request[i][j] <= w[j])
        canbeprocessed = 1;
      else {
        canbeprocessed = 0;
        break;
      }
    }
```

```
  if (canbeprocessed) {
        mark[i] = 1;

        for (j = 0; j < nr; j++)
          w[j] += alloc[i][j];
      }
    }
  }

  //checking for unmarked processes
  int deadlock = 0;
  for (i = 0; i < np; i++)
  if (mark[i] != 1)
     deadlock = 1;

  if (deadlock){
    printf("\n Deadlock detected");
  }
  else
    printf("\n No Deadlock possible");
printf("\n--------------------------------------------------\n");
}
```

OUTPUT:


gini@gini: ~/Practicals/OS_LAB_4
gini@gini:~/Practicals/OS_LAB_4$ gcc deadlock_detection.c
gini@gini:~/Practicals/OS_LAB_4$ ./a.out
**********DEADLOCK DETECTION**********
Enter the no of process : 4
Enter the no of resources : 5
Total Amount of the Resource R1 : 2
Total Amount of the Resource R2 : 1
Total Amount of the Resource R3 : 1
Total Amount of the Resource R4 : 2
Total Amount of the Resource R5 : 1
Enter the request matrix :
0 1 0 0 1
0 0 1 0 1
0 0 0 0 1
1 0 1 0 1
Enter the allocation matrix :
1 0 1 1 0
1 1 0 0 0
0 0 0 1 0
0 0 0 0 0
Deadlock detected

B.] DEADLOCK AVOIDANCE

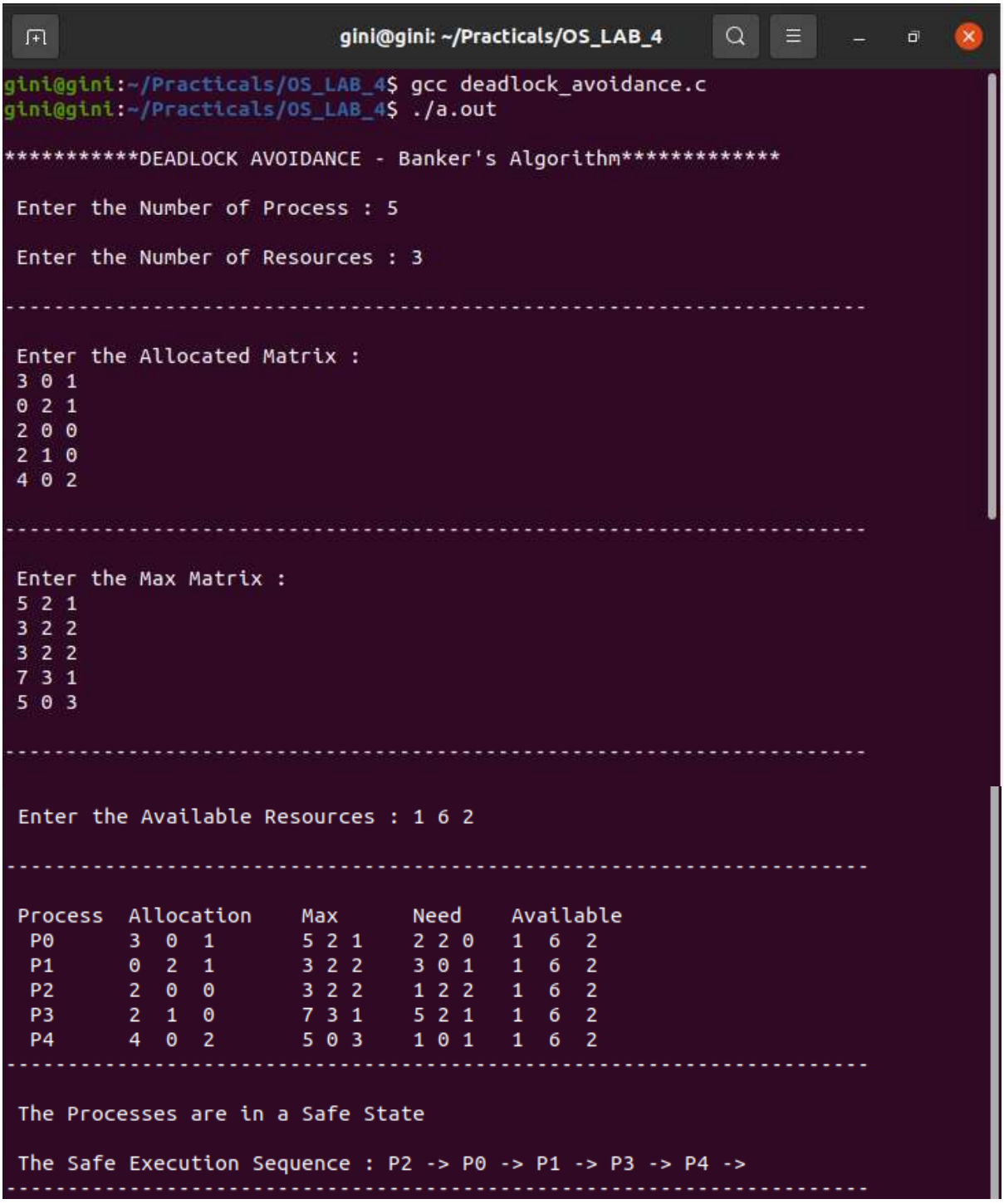
CODE:

```
#include<stdio.h>

int main() {
  int p, res;
  printf("\n**********DEADLOCK AVOIDANCE - Banker's
Algorithm**********\n");
  printf("\n Enter the Number of Process : ");
  scanf("%d", & p);
  printf("\n Enter the Number of Resources : ");
  scanf("%d", & res);
  printf("\n----------------------------------------------------------------------\n");

  int allocated[p][res], need[p][res], max[p][res], status[p], seq[p], available[1][res], flag = 0;
  printf("\n Enter the Allocated Matrix : \n");
  for (int i = 0; i < p; i++) {
    for (int j = 0; j < res; j++) {
      scanf("%d", & allocated[i][j]);
    }
  }

  printf("\n----------------------------------------------------------------------\n");

  printf("\n Enter the Max Matrix : \n");
  for (int i = 0; i < p; i++) {
    for (int j = 0; j < res; j++) {
      scanf("%d", & max[i][j]);
    }
  }

  printf("\n----------------------------------------------------------------------\n");
  for (int i = 0; i < p; i++) {
    for (int j = 0; j < res; j++) {
      need[i][j] = max[i][j] - allocated[i][j];
    }
  }

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

```
printf("\n---------------------------------------------------------------------\n");
printf("\n Process\t Allocation\t\tMax\t\t Need\t\t Available\t");
for (int i = 0; i < p; i++) {
  printf("\n  P%d\t\t ", i);
  for (int j = 0; j < res; j++) {
    printf(" %d ", allocated[i][j]);
  }

  printf("\t\t");
  for (int j = 0; j < res; j++) {
    printf("%d ", max[i][j]);
  }

  printf("\t\t");
  for (int j = 0; j < res; j++) {
    printf(" %d", max[i][j] - allocated[i][j]);
  }

  printf("\t\t");

  for (int j = 0; j < res; j++)
    printf(" %d ", available[0][j]);

}
printf("\n---------------------------------------------------------------------\n");

for (int i = 0; i < p; i++) {
  status[i] = 0;
}

int counter = 0;
while (counter < p) {
  int st = 0;
  int processid = 9999;
  for (int i = 0; i < p; i++) {
    for (int j = 0; j < res; j++) {
      if (available[0][j] >= need[i][j] && status[i] == 0) {
        st = 1;
        processid = i;
      } else {
        st = 0;
        break;
      }
    }
```

```
    if (st == 1) {
      break;
    } else {
      continue;
    }
  }

  if (processid == 9999) {
    printf(" The Processess are in Deadlock ");
    printf("\n---------------------------------------------------------------------\n");
    flag = 1;
    break;
  }

  seq[counter] = processid;
  status[processid] = 1;
  counter++;
  for (int j = 0; j < res; j++) {
    available[0][j] = available[0][j] + allocated[processid][j];
  }

 }

 if (flag == 0) {
  printf("\n The Processes are in a Safe State");
  printf("\n\n The Safe Execution Sequence : ");
  for (int i = 0; i < p; i++) {
    printf("P%d -> ", seq[i]);
  }
  printf("\n---------------------------------------------------------------------\n");
 }

 return 0;
}
```

OUTPUT:


gini@gini: ~/Practicals/OS_LAB_4
gini@gini:~/Practicals/OS_LAB_4$ gcc deadlock_avoidance.c
gini@gini:~/Practicals/OS_LAB_4$ ./a.out
**********DEADLOCK AVOIDANCE - Banker's Algorithm**********
Enter the Number of Process : 5
Enter the Number of Resources : 3
Enter the Allocated Matrix :
3 0 1
0 2 1
2 0 0
2 1 0
4 0 2
Enter the Max Matrix :
5 2 1
3 2 2
3 2 2
7 3 1
5 0 3
Enter the Available Resources : 1 6 2
Process Allocation Max Need Available
P0 3 0 1 5 2 1 2 2 0 1 6 2
P1 0 2 1 3 2 2 3 0 1 1 6 2
P2 2 0 0 3 2 2 1 2 2 1 6 2
P3 2 1 0 7 3 1 5 2 1 1 6 2
P4 4 0 2 5 0 3 1 0 1 1 6 2
The Processes are in a Safe State
The Safe Execution Sequence : P2 -> P0 -> P1 -> P3 -> P4 ->