OPERATING SYSTEMS LAB
DIGITAL ASSIGNMENT-2

AADITYA ROSHAN
22BIT0250

(a) Write a Program to implement banker's algorithm for finding a Safe Sequence.

CODE:-

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

int main() {
  int i, j, k;
  int processes, resources;

  printf("Enter the number of processes: ");
  scanf("%d", &processes);
  printf("Enter the number of resources: ");
  scanf("%d", &resources);

  int available[resources];
  int max[processes][resources];
  int allocation[processes][resources];
  int need[processes][resources];
  bool finish[processes];

  printf("Enter the available resources for each resource type:\n");
  for (i = 0; i < resources; i++) {
    printf("Resource %d: ", i + 1);
    scanf("%d", &available[i]);
  }

  printf("Enter the maximum demand for each process:\n");
  for (i = 0; i < processes; i++) {
    printf("Process %d:\n", i + 1);
    for (j = 0; j < resources; j++) {
      printf("Maximum demand for Resource %d: ", j + 1);
      scanf("%d", &max[i][j]);
    }
  }

  printf("Enter the allocated resources for each process:\n");
  for (i = 0; i < processes; i++) {
    printf("Process %d:\n", i + 1);
    for (j = 0; j < resources; j++) {
      printf("Allocated resources for Resource %d: ", j + 1);
      scanf("%d", &allocation[i][j]);
    }
  }

  // Calculate the need matrix
  for (i = 0; i < processes; i++) {
    for (j = 0; j < resources; j++) {
      need[i][j] = max[i][j] - allocation[i][j];
```

```
    }
    finish[i] = false;  // Initialize finish array to false
  }

  int safeSequence[processes];
  int work[resources];
  for (i = 0; i < resources; i++) {
    work[i] = available[i];
  }

  int count = 0;
  while (count < processes) {
    bool found = false;
    for (i = 0; i < processes; i++) {
      if (!finish[i]) {
        bool canAllocate = true;
        for (j = 0; j < resources; j++) {
          if (need[i][j] > work[j]) {
            canAllocate = false;
            break;
          }
        }

        if (canAllocate) {
          for (k = 0; k < resources; k++) {
            work[k] += allocation[i][k];
          }
          safeSequence[count++] = i;
          finish[i] = true;
          found = true;
        }
      }
    }

    if (!found) {
      printf("System is not in a safe state.\n");
      return -1;
    }
  }

  // Output the safe sequence
  printf("System is in a safe state.\nSafe sequence is: ");
  for (i = 0; i < processes; i++) {
    printf("P%d ", safeSequence[i] + 1);
  }
  printf("\n");

  return 0;
}
```

OUTPUT:-

```
aaditya@AadityaPC:~$ gcc lab_da2a.c -o lab_da2a
aaditya@AadityaPC:~$ ./lab_da2a
Enter the number of processes: 4
Enter the number of resources: 3
Enter the available resources for each resource type:
Resource 1: 3
Resource 2: 3
Resource 3: 2
Enter the maximum demand for each process:
Process 1:
Maximum demand for Resource 1: 7
Maximum demand for Resource 2: 5
Maximum demand for Resource 3: 3
Process 2:
Maximum demand for Resource 1: 3
Maximum demand for Resource 2: 2
Maximum demand for Resource 3: 2
Process 3:
Maximum demand for Resource 1: 9
Maximum demand for Resource 2: 0
Maximum demand for Resource 3: 2
Process 4:
Maximum demand for Resource 1: 2
Maximum demand for Resource 2: 2
Maximum demand for Resource 3: 2
Enter the allocated resources for each process:
Process 1:
Allocated resources for Resource 1: 0
Allocated resources for Resource 2: 1
Allocated resources for Resource 3: 0
Process 2:
Allocated resources for Resource 1: 2
Allocated resources for Resource 2: 0
Allocated resources for Resource 3: 0
Process 3:
Allocated resources for Resource 1: 3
Allocated resources for Resource 2: 0
Allocated resources for Resource 3: 2
Process 4:
Allocated resources for Resource 1: 2
Allocated resources for Resource 2: 1
Allocated resources for Resource 3: 1
System is in a safe state.
Safe sequence is: P2 P4 P1 P3
```

(b) Write a Program to implement banker's algorithm for Resource Request.

CODE:-

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

int main() {
  int i, j;
  int processes, resources;

  // Input number of processes and resources
  printf("Enter the number of processes: ");
  scanf("%d", &processes);
  printf("Enter the number of resources: ");
  scanf("%d", &resources);

  // Declare arrays for available, max, allocation, and need
  int available[resources];
  int max[processes][resources];
  int allocation[processes][resources];
  int need[processes][resources];

  // Input available resources
  printf("Enter the available resources for each resource type:\n");
  for (i = 0; i < resources; i++) {
    printf("Resource %d: ", i + 1);
    scanf("%d", &available[i]);
  }

  // Input maximum demand for each process
  printf("Enter the maximum demand for each process:\n");
  for (i = 0; i < processes; i++) {
    printf("Process %d:\n", i + 1);
    for (j = 0; j < resources; j++) {
      printf("Maximum demand for Resource %d: ", j + 1);
      scanf("%d", &max[i][j]);
    }
  }

  // Input allocated resources for each process
  printf("Enter the allocated resources for each process:\n");
  for (i = 0; i < processes; i++) {
    printf("Process %d:\n", i + 1);
    for (j = 0; j < resources; j++) {
      printf("Allocated resources for Resource %d: ", j + 1);
      scanf("%d", &allocation[i][j]);
    }
  }

  // Calculate need matrix (max - allocation)
  for (i = 0; i < processes; i++) {
    for (j = 0; j < resources; j++) {
      need[i][j] = max[i][j] - allocation[i][j];
    }
  }

  // Input the process that is making the request
```

```
int process_id;
printf("Enter the process number making the request (0 to %d): ", processes - 1);
scanf("%d", &process_id);

// Input the resource request for the process
int request[resources];
printf("Enter the resource request for process %d:\n", process_id);
for (i = 0; i < resources; i++) {
  printf("Requested Resource %d: ", i + 1);
  scanf("%d", &request[i]);
}

// Check if request is more than need
for (i = 0; i < resources; i++) {
  if (request[i] > need[process_id][i]) {
    printf("Error: Process has exceeded its maximum claim.\n");
    return -1;
  }
}

// Check if request is more than available
for (i = 0; i < resources; i++) {
  if (request[i] > available[i]) {
    printf("Process %d must wait, resources are not available.\n", process_id);
    return -1;
  }
}

// Temporarily allocate the resources
for (i = 0; i < resources; i++) {
  available[i] -= request[i];
  allocation[process_id][i] += request[i];
  need[process_id][i] -= request[i];
}

// Safety check (Banker's Algorithm)
int work[resources];
bool finish[processes];
for (i = 0; i < resources; i++) {
  work[i] = available[i];
}
for (i = 0; i < processes; i++) {
  finish[i] = false;
}

int count = 0, safeSequence[processes];
while (count < processes) {
  bool found = false;
  for (i = 0; i < processes; i++) {
    if (!finish[i]) {
      bool canAllocate = true;
      for (j = 0; j < resources; j++) {
        if (need[i][j] > work[j]) {
          canAllocate = false;
          break;
        }
      }
```

```
            if (canAllocate) {
                for (j = 0; j < resources; j++) {
                    work[j] += allocation[i][j];
                }
                safeSequence[count++] = i;
                finish[i] = true;
                found = true;
            }
        }
    }
    if (!found) {
        // Rollback the allocation
        for (i = 0; i < resources; i++) {
            available[i] += request[i];
            allocation[process_id][i] -= request[i];
            need[process_id][i] += request[i];
        }
        printf("The system is not in a safe state. Request cannot be granted.\n");
        return -1;
    }
  }

  // Output the safe sequence
  printf("The system is in a safe state.\nSafe sequence is: ");
  for (i = 0; i < processes; i++) {
    printf("P%d ", safeSequence[i] + 1);
  }
  printf("\n");

  return 0;
}
```

OUTPUT:-

```
aaditya@AadityaPC:~$ ./lab_da2b
Enter the number of processes: 5
Enter the number of resources: 3
Enter the available resources for each resource type:
Resource 1: 3
Resource 2: 3
Resource 3: 2
Enter the maximum demand for each process:
Process 1:
Maximum demand for Resource 1: 7
Maximum demand for Resource 2: 5
Maximum demand for Resource 3: 3
Process 2:
Maximum demand for Resource 1: 3
Maximum demand for Resource 2: 2
Maximum demand for Resource 3: 2
Process 3:
Maximum demand for Resource 1: 9
Maximum demand for Resource 2: 0
Maximum demand for Resource 3: 2
Process 4:
Maximum demand for Resource 1: 2
Maximum demand for Resource 2: 2
Maximum demand for Resource 3: 2
Process 5:
Maximum demand for Resource 1: 4
Maximum demand for Resource 2: 3
Maximum demand for Resource 3: 3
Enter the allocated resources for each process:
Process 1:
Allocated resources for Resource 1: 0
Allocated resources for Resource 2: 1
Allocated resources for Resource 3: 0
Process 2:
Allocated resources for Resource 1: 2
Allocated resources for Resource 2: 0
Allocated resources for Resource 3: 0
Process 3:
Allocated resources for Resource 1: 3
Allocated resources for Resource 2: 0
Allocated resources for Resource 3: 2
Process 4:
Allocated resources for Resource 1: 2
Allocated resources for Resource 2: 1
Allocated resources for Resource 3: 1
Process 5:
Allocated resources for Resource 1: 0
Allocated resources for Resource 2: 0
Allocated resources for Resource 3: 2
Enter the process number making the request (0 to 4): 1
Enter the resource request for process 1:
Requested Resource 1: 1
Requested Resource 2: 0
Requested Resource 3: 2
The system is in a safe state.
Safe sequence is: P2 P4 P5 P1 P3
```