Bankers Algorithm

program

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

typedef struct {
   int res[10];
   int max[10][10];
   int alloc[10][10];
   int need[10][10];
   int avail[10];
} bankers;

bankers input(bankers b, int p, int r) {
   int i, j;
   printf("Enter the Number of Total resources in the system:\n");
   for (i = 0; i < r; i++) {
      printf("for resource R%d: ", i);
      scanf("%d", &b.res[i]);
   }

   printf("Enter the resources allocated to each process (allocated resource table):\n");
   for (i = 0; i < p; i++) {
      //printf("For Process P%d:\n", i);
      for (j = 0; j < r; j++) {
         //printf("for resource R%d: ", j);
         scanf("%d", &b.alloc[i][j]);
      }
   }

   printf("Enter the Maximum resources needed by each process (Max claim table):\n");
   for (i = 0; i < p; i++) {
      //printf("For Process P%d:\n", i);
      for (j = 0; j < r; j++) {
         //printf("for resource R%d: ", j);
         scanf("%d", &b.max[i][j]);
      }
   }

   for (i = 0; i < p; i++) {
      for (j = 0; j < r; j++) {
         b.need[i][j] = b.max[i][j] - b.alloc[i][j];
      }
   }

   for (i = 0; i < r; i++) {
      b.avail[i] = b.res[i];
      for (j = 0; j < p; j++) {
         b.avail[i] -= b.alloc[j][i];
      }
```

```
    }

    return b;
}

int safety(bankers b, int p, int r) {
    int i, j, flag = 0, target = 0;
    int finish[10] = {0};
    int work[10];
    int safeSequence[10];

    for (i = 0; i < r; i++) {
        work[i] = b.avail[i];
    }

    while (target < p) {
        flag = 0;
        for (i = 0; i < p; i++) {
            if (!finish[i]) {
                int can_allocate = 1;
                for (j = 0; j < r; j++) {
                    if (b.need[i][j] > work[j]) {
                        can_allocate = 0;
                        break;
                    }
                }

                if (can_allocate) {
                    finish[i] = 1;
                    safeSequence[target++] = i;
                    for (j = 0; j < r; j++) {
                        work[j] += b.alloc[i][j];
                    }
                    flag = 1;
                    break;
                }
            }
        }

        if (!flag) {
            printf("System is in an unsafe state! Deadlock may occur.\n");
            return 0;
        }
    }

    printf("System is in a safe state! Safe Sequence: ");
    for (i = 0; i < p; i++) {
        printf("P%d ", safeSequence[i]);
    }
    printf("\n");
    return 1;
}
```

```
bankers request(bankers b, int p, int r) {
    int process_id, i, request[10];

    printf("Enter the process number (0 to %d): ", p - 1);
    scanf("%d", &process_id);

    if (process_id < 0 || process_id >= p) {
        printf("Invalid process ID!\n");
        return b;
    }

    printf("Enter the resource request for process P%d:\n", process_id);
    for (i = 0; i < r; i++) {
        printf("Resource R%d: ", i);
        scanf("%d", &request[i]);
    }

    for (i = 0; i < r; i++) {
        if (request[i] > b.need[process_id][i]) {
            printf("Error: Process has exceeded its maximum claim!\n");
            return b;
        }
        if (request[i] > b.avail[i]) {
            printf("Error: Insufficient resources available!\n");
            return b;
        }
    }

    for (i = 0; i < r; i++) {
        b.avail[i] -= request[i];
        b.alloc[process_id][i] += request[i];
        b.need[process_id][i] -= request[i];
    }

    if (safety(b, p, r)) {
        printf("Resource request granted!\n");
    } else {
        for (i = 0; i < r; i++) {
            b.avail[i] += request[i];
            b.alloc[process_id][i] -= request[i];
            b.need[process_id][i] += request[i];
        }
        printf("Resource request denied! System would enter an unsafe state.\n");
    }

    return b;
}


int main() {
    bankers b;
```

```
    int p, r, choice;

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

    b = input(b, p, r);

    while (1) {
        printf("\nMenu:\n");
        printf("1. Check Safe State (Banker's Algorithm)\n");
        printf("2. Request Resources\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                safety(b, p, r);
                break;
            case 2:
                b = request(b, p, r);
                break;
            case 3:
                printf("Exiting the program.\n");
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}
```

```
gokul@gokul-ThinkPad-T460s:~/S4/OS/LabCycle/EXP9_Bankers Algorithm$ gedit banker.c
gokul@gokul-ThinkPad-T460s:~/S4/OS/LabCycle/EXP9_Bankers Algorithm$ gcc banker.c -o banker.out
gokul@gokul-ThinkPad-T460s:~/S4/OS/LabCycle/EXP9_Bankers Algorithm$ ./banker.out
Enter number of processes: 5
Enter number of resources: 3
Enter the Number of Total resources in the system:
for resource R0: 10
for resource R1: 5
for resource R2: 7
Enter the resources allocated to each process (allocated resource table):
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter the Maximum resources needed by each process (Max claim table):
7 5 3
3 2 2
9 0 2
2 2 2
4 3
3

Menu:
1. Check Safe State (Banker's Algorithm)
2. Request Resources
3. Exit
Enter your choice: 1
System is in a safe state! Safe Sequence: P1 P3 P0 P2 P4

Menu:
1. Check Safe State (Banker's Algorithm)
2. Request Resources
3. Exit
Enter your choice: 2
Enter the process number (0 to 4): 1
Enter the resource request for process P1:
Resource R0: 1
Resource R1: 0
Resource R2: 2
System is in a safe state! Safe Sequence: P1 P3 P0 P2 P4
Resource request granted!

Menu:
1. Check Safe State (Banker's Algorithm)
2. Request Resources
3. Exit
Enter your choice:
```