# UCS 1411 - Operating Systems Lab

**Exercise 7 - Implementation of Banker's algorithm (deadlock avoidance)**

Mahesh Bharadwaj K - 185001089

## 1 Develop a C program to implement the Banker's algorithm for deadlock avoidance

### Program

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

#define MAX 5

typedef struct Resource
{
    char name;
    unsigned short int qty;
} Resource;

typedef struct Process
{
    int pid;
    Resource max[MAX];
    Resource alloc[MAX];
    Resource need[MAX];
    unsigned completed : 1;
} Process;

void ReadData(int *const, Process *const, int *const, Resource *const);
void PrintData(const int, const Process *const, const int, const Resource *const);
int SafeSequence(const int, const Process *const, const int, const Resource *const);
void RequestAllocation(const int, Process *const, const int, Resource *const);

int main()
{

    int n_process = 0,
        n_resources = 0,
        choice = -1;


    Process p[MAX * 2];
    Resource avail[MAX];

    while (1)
    {
        printf("\t\t\tBANKERS ALGORITHM\n");
        printf(" 1 - Read Data\n");
        printf(" 2 - Print Data\n");
        printf(" 3 - Safe Sequence\n");
```

```
        printf(" 4 - Request for Resource\n");
        printf(" 0 - exit\n");
        printf(" ---------------------------\n");
        printf(" Enter your choice : ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
            ReadData(&n_process, p, &n_resources, avail);
            break;
        case 2:
            PrintData(n_process, p, n_resources, avail);
            break;
        case 3:
            SafeSequence(n_process, p, n_resources, avail);
            break;
        case 4:
            RequestAllocation(n_process, p, n_resources, avail);
            break;
        case 0:
            exit(0);
        default:
            printf(" Invalid Option!\n");
            break;
        }
        printf("\n\n");
    }

}

void ReadData(int *const n_process, Process *const arr, int *const n_resources,
↪  Resource *const avail)
{
    printf(" Enter the Number of Processes: ");
    scanf("%d", n_process);
    printf(" Enter the Number of Resources: ");
    scanf("%d", n_resources);
    getchar();

    for (int i = 0; i < *n_resources; i++)
    {
        printf(" Enter the name of resource & available: ");
        scanf("%c %hd", &avail[i].name, &avail[i].qty);
        getchar();
    }

    for (int i = 0; i < *n_process; i++)
    {
        arr[i].completed = 0;
        printf("Enter Process ID, Max Required, Allocated: ");
        scanf("%d", &arr[i].pid);

        for (int j = 0; j < *n_resources; j++)
            scanf("%hd", &arr[i].max[j].qty);

        for (int j = 0; j < *n_resources; j++)
        {
```

```
            scanf("%hd", &arr[i].alloc[j].qty);
            arr[i].need[j].qty = arr[i].max[j].qty - arr[i].alloc[j].qty;
        }
    }
}



void PrintData(const int n_process, const Process *const arr, const int n_resources,
↪  const Resource *const avail)
{

    printf("\n");
    printf("
    ↪  +-----+----------------+---------------+----------------+---------------+\n");
    printf(" | PID |   Allocated    |    Needed     |    Maximum     |   Available
    ↪  |\n");
    printf(" |     |  ");

    for (int i = 0; i < n_resources; i++)
        printf("%c  ", avail[i].name);

    for (int i = n_resources * 3; i < strlen(" Allocated  "); i++)
        printf(" ");

    printf("  |  ");

    for (int i = 0; i < n_resources; i++)
        printf("%c  ", avail[i].name);

    for (int i = n_resources * 3; i < strlen("  Needed   "); i++)
        printf(" ");

    printf("  |  ");

    for (int i = 0; i < n_resources; i++)
        printf("%c  ", avail[i].name);

    for (int i = n_resources * 3; i < strlen("  Maximum   "); i++)
        printf(" ");

    printf("  |  ");

    for (int i = 0; i < n_resources; i++)
        printf("%c  ", avail[i].name);

    for (int i = n_resources * 3; i < strlen(" Available "); i++)
        printf(" ");

    printf("  |\n");
    printf("
    ↪  +-----+----------------+---------------+----------------+---------------+\n");

    for (int k = 0; k < n_process; k++)
    {
        printf(" | P%-2d |  ", arr[k].pid);
        for (int i = 0; i < n_resources; i++)
            printf("%-2d ", arr[k].alloc[i].qty);
```

```
        for (int i = n_resources * 3; i < strlen(" Allocated  "); i++)
            printf(" ");

        printf("  |  ");

        for (int i = 0; i < n_resources; i++)
            printf("%-2d ", arr[k].need[i].qty);

        for (int i = n_resources * 3; i < strlen("  Needed   "); i++)
            printf(" ");

        printf("  |  ");

        for (int i = 0; i < n_resources; i++)
            printf("%-2d ", arr[k].max[i].qty);

        for (int i = n_resources * 3; i < strlen("  Maximum   "); i++)
            printf(" ");

        printf("  |  ");

        if (k == 0)
        {
            for (int i = 0; i < n_resources; i++)
                printf("%-2d ", avail[i].qty);

            for (int i = n_resources * 3; i < strlen(" Available "); i++)
                printf(" ");
        }
        else
            printf("           ");

        printf("  |\n");
    }
    printf("
    ↪  +-----+---------------+--------------+---------------+--------------+\n");
}

int findProcess(const int n_process, const Process *const arr, const int n_resources,
↪  const Resource *const avail, const int index)
{
    int flag = 0;
    for (int i = (index + 1) % n_process; i != index; i = (i + 1) % n_process)
    {
        flag = 0;

        if (arr[i].completed)
            continue;

        for (int j = 0; j < n_resources; j++)
        {
            if (arr[i].need[j].qty > avail[j].qty)
            {
                flag = 1;
                break;
            }
        }
```

```
        if (!flag)
            return i;

        if (index == -1 && i == n_process - 1)
            break;
    }
    return -1;
}

int SafeSequence(const int n_process, const Process *const arr, const int n_resources,
↪   const Resource *const avail)
{
    //Creating a copy of the processes
    Process p[MAX * 2];
    Resource avail_copy[MAX];
    for (int i = 0; i < n_process; i++)
        p[i] = arr[i];

    for (int i = 0; i < n_resources; i++)
        avail_copy[i] = avail[i];

    int completed = 0;
    int index = -1;

    int sequence[MAX];

    while (completed < n_process)
    {
        index = findProcess(n_process, p, n_resources, avail_copy, index);

        if (index == -1)
            break;

        sequence[completed++] = p[index].pid;
        p[index].completed = 1;

        for (int i = 0; i < n_resources; i++)
        {
            avail_copy[i].qty += p[index].alloc[i].qty;
        }
        printf("\n");
    }

    //All Processes done, ie safe sequence exists
    if (completed == n_process)
    {
        printf(" Safe Sequence Exists!\n");
        printf(" < ");
        for (int i = 0; i < n_process; i++)
            printf("P%-2d ", sequence[i]);
        printf(">\n");
        return 1;
    }
    else
        printf(" No Safe Sequence Found!");
    return 0;
}
```

```
void RequestAllocation(const int n_process, Process *const arr, const int n_resources,
↪  Resource *const avail)
{
    int pid;
    Resource request[MAX];

    printf(" PID & Enter the request vector: ");
    scanf("%d", &pid);
    for (int i = 0; i < n_resources; i++)
        scanf("%hd", &request[i].qty);

    for (int i = 0; i < n_resources; i++)
        if (request[i].qty > avail[i].qty || request[i].qty > arr[pid].need[i].qty)
        {
            printf(" Invalid Request! Cannot be granted!\n");
            return;
        }

    Resource old_alloc[MAX], old_need[MAX], old_avail[MAX];

    for (int i = 0; i < n_resources; i++)
    {
        old_alloc[i].qty = arr[pid].alloc[i].qty;
        old_need[i].qty = arr[pid].need[i].qty;
        old_avail[i].qty = avail[i].qty;
        avail[i].qty -= request[i].qty;
        arr[pid].need[i].qty -= request[i].qty;
        arr[pid].alloc[i].qty += request[i].qty;
    }

    if (SafeSequence(n_process, arr, n_resources, avail))
        printf(" Safe Sequence exists & Hence Request granted!\n");
    else
        printf(" Request Cannot be granted as safe sequence doesn't exist!\n");

    for (int i = 0; i < n_resources; i++)
    {
        arr[pid].alloc[i].qty = old_alloc[i].qty;
        arr[pid].need[i].qty = old_need[i].qty;
        avail[i].qty = old_avail[i].qty;
    }
}
```

---

## Output

```
BANKERS ALGORITHM
 1 - Read Data
 2 - Print Data
 3 - Safe Sequence
 4 - Request for Resource
 0 - exit
 ---------------------------
 Enter your choice : 1
 Enter the Number of Processes: 5
 Enter the Number of Resources: 3
 Enter the name of resource & available: A 3
```

```
 Enter the name of resource & available: B 3
 Enter the name of resource & available: C 2
Enter Process ID, Max Required, Allocated: 0 7 5 3 0 1 0
Enter Process ID, Max Required, Allocated: 1 3 2 2 2 0 0
Enter Process ID, Max Required, Allocated: 2 9 0 2 3 0 2
Enter Process ID, Max Required, Allocated: 3 2 2 2 2 1 1
Enter Process ID, Max Required, Allocated: 4 4 3 3 0 0 2


BANKERS ALGORITHM
 1 - Read Data
 2 - Print Data
 3 - Safe Sequence
 4 - Request for Resource
 0 - exit
 ---------------------------
 Enter your choice : 2

 +-----+----------------+---------------+----------------+---------------+
 | PID |   Allocated    |    Needed     |    Maximum     |   Available   |
 |     |  A  B  C       |  A  B  C      |  A  B  C       |  A  B  C      |
 +-----+----------------+---------------+----------------+---------------+
 | P0  |  0  1  0       |  7  4  3      |  7  5  3       |  3  3  2      |
 | P1  |  2  0  0       |  1  2  2      |  3  2  2       |               |
 | P2  |  3  0  2       |  6  0  0      |  9  0  2       |               |
 | P3  |  2  1  1       |  0  1  1      |  2  2  2       |               |
 | P4  |  0  0  2       |  4  3  1      |  4  3  3       |               |
 +-----+----------------+---------------+----------------+---------------+


BANKERS ALGORITHM
 1 - Read Data
 2 - Print Data
 3 - Safe Sequence
 4 - Request for Resource
 0 - exit
 ---------------------------
 Enter your choice : 3




 Safe Sequence Exists!
 < P1  P3  P4  P0  P2  >


BANKERS ALGORITHM
 1 - Read Data
 2 - Print Data
 3 - Safe Sequence
 4 - Request for Resource
 0 - exit
 ---------------------------
 Enter your choice : 4
 PID & Enter the request vector: 1 1 0 2
```

```
 Safe Sequence Exists!
 < P1  P3  P4  P0  P2  >
 Safe Sequence exists & Hence Request granted!


BANKERS ALGORITHM
 1 - Read Data
 2 - Print Data
 3 - Safe Sequence
 4 - Request for Resource
 0 - exit
 ---------------------------
 Enter your choice : 4
 PID & Enter the request vector: 1 3 3 2
 Invalid Request! Cannot be granted!


BANKERS ALGORITHM
 1 - Read Data
 2 - Print Data
 3 - Safe Sequence
 4 - Request for Resource
 0 - exit
 ---------------------------
 Enter your choice : 0
```

---