****

**LAB EXERCISE 6**

**Implementation of Producer/Consumer Problem using Semaphores**

**Submission Date:21-04-2022**

Name: Jayannthan P T

Dept: CSE ‘A’

Roll No.: 205001049

1. Assignment 1: Develop a C program to implement Banker’s algorithm for deadlock avoidance with multiple instances of resource types

**Algorithm:**

1. Get choice from user
2. If choice is equal to 1, then ask for data input
3. If choice is equal to 2, then print the data
4. If choice is equal to 3, then call bankers algorithm to execute
5. If choice is equal to 4 then call for resource request
6. Else exit

Algorithm for data input:

1. Get no of process, no of resources, available instances, maximum required matrix, allotted instances from user
2. Calculate need matrix by subtracting allocation matrix from maximum required matrix

Algorithm for bankers algorithm:

1. Set ind to 0
2. For i from 0 to no of process times
   1. If f[i] is equal to 0
      1. Set flag = 0
   2. For j no of resources times
      1. If need[I,j]greater than available[j] then set flag = 1
   3. If flag equal to 0 then
      1. Set safeseq[ind]=i
      2. Increment ind
      3. Set available [j] equal to sum of available[j] and allocstion[I,j]
      4. Set f[i] is equal to 1
3. Set flag equal to 1
4. For i from 0 to no of process
   1. If f[i] is equal to 0 then print not a safe sequence and exit
   2. If flag is equal to 1 then print safe sequence

Algorithm for Resource Request:

1. Get process no from user to allocate
2. Get resource vector
3. Check if allocation[i] is greater than available[i] then exit
4. Else set available equal to available[i] minus allocationvector[i] and allocation[ind][i] equal to sum of allocation[ind][i] and allocationvector[i]
5. Now call bankers algorithm

**Code:**

*// Assignment 1: Develop a C program to implement Banker’s algorithm for deadlock avoidance with multiple instances of resource types*

#include <stdio.h>

#include <string.h>

#define max 100

typedef **struct** bankersdata

{

**int** no\_of\_process;

**int** no\_of\_resources;

**char** process\_name[max][5];

**char** resources\_name[max][5];

**int** available\_instance[max];   *// available\_instance[no\_of\_resources]*

**int** max\_req[max][max];         *// max\_req[no\_of\_process][no\_of\_resources];*

**int** allocation[max][max];      *// allocation[no\_of\_process][no\_of\_resources];*

**int** f[max], safesequence[max]; *// f[no\_of\_process],safesequence[no\_of\_process]*

**int** need[max][max];

} bankersdata;

**void** printdata(bankersdata \*bk)

{

    printf("Pid\tAlloc\tMax  \tNeed \tAvail\t\n--\t");

    for (**int** i = 0; i < bk->no\_of\_resources; i++)

    {

        printf("%s ", bk->resources\_name[i]);

    }

    printf("\t");

    for (**int** i = 0; i < bk->no\_of\_resources; i++)

    {

        printf("%s ", bk->resources\_name[i]);

    }

    printf("\t");

    for (**int** i = 0; i < bk->no\_of\_resources; i++)

    {

        printf("%s ", bk->resources\_name[i]);

    }

    printf("\t");

    for (**int** i = 0; i < bk->no\_of\_resources; i++)

    {

        printf("%s ", bk->resources\_name[i]);

    }

*// printf("\n");*

    printf("\t");

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

    for (**int** i = 0; i < bk->no\_of\_process; i++)

    {

        printf("%s ", bk->process\_name[i]);

        for (**int** j = 0; j < bk->no\_of\_resources; j++)

        {

            printf(" %d", bk->allocation[i][j]);

        }

        printf("\t");

        for (**int** j = 0; j < bk->no\_of\_resources; j++)

        {

            printf(" %d", bk->max\_req[i][j]);

        }

        printf("\t");

        for (**int** j = 0; j < bk->no\_of\_resources; j++)

        {

            printf(" %d", bk->need[i][j]);

        }

        printf("\t");

        if (i == 0)

        {

            for (**int** j = 0; j < bk->no\_of\_resources; j++)

            {

                printf(" %d", bk->available\_instance[j]);

            }

        }

        printf("\t\n");

    }

}

**void** bankersalgo(bankersdata \*bk)

{

    printdata(bk);

**int** ind = 0;

**int** y = 0;

    for (**int** k = 0; k < 5; k++)

    {

        for (**int** i = 0; i < bk->no\_of\_process; i++)

        {

            if (bk->f[i] == 0)

            {

**int** flag = 0;

                for (**int** j = 0; j < bk->no\_of\_resources; j++)

                {

                    if (bk->need[i][j] > bk->available\_instance[j])

                    {

                        flag = 1;

                        break;

                    }

                }

                if (flag == 0)

                {

                    bk->safesequence[ind] = i;

                    ind++;

                    for (**int** j = 0; j < bk->no\_of\_resources; j++)

                    {

                        bk->available\_instance[j] += bk->allocation[i][j];

                    }

                    bk->f[i] = 1;

                }

            }

        }

    }

**int** flag = 1;

    for (**int** i = 0; i < bk->no\_of\_process; i++)

    {

        if (bk->f[i] == 0)

        {

            flag = 0;

            printf("NOT A SAFE SYSTEM");

            break;

        }

    }

    if (flag == 1)

    {

        printf("SAFE SEQUENCE\n");

        for (**int** i = 0; i < bk->no\_of\_process - 1; i++)

        {

            printf(" %s ->", bk->process\_name[bk->safesequence[i]]);

        }

        printf(" %s", bk->process\_name[bk->safesequence[bk->no\_of\_process - 1]]);

    }

    printdata(bk);

}

bankersdata getdata()

{

    bankersdata bk;

    printf("\nEnter no of process:");

    scanf("%d", &bk.no\_of\_process);

    printf("\nEnter process ids:\n");

    for (**int** i = 0; i < bk.no\_of\_process; i++)

    {

        printf("process name of process %d:", i + 1);

        scanf(" %s", &bk.process\_name[i]);

    }

    printf("\nEnter no of resources:");

    scanf("%d", &bk.no\_of\_resources);

    printf("\nEnter resource ids:\n");

    for (**int** i = 0; i < bk.no\_of\_resources; i++)

    {

        printf("resource name of resource %d:", i + 1);

        scanf(" %s", &bk.resources\_name[i]);

    }

    printf("\nEnter available instances:\n");

    for (**int** i = 0; i < bk.no\_of\_resources; i++)

    {

        printf("available instances of resource %s:", bk.resources\_name[i]);

        scanf(" %d", &bk.available\_instance[i]);

    }

    printf("\nEnter Maximum requirement:\n");

    for (**int** i = 0; i < bk.no\_of\_process; i++)

    {

        printf("Maximum requirement for process %s:", bk.process\_name[i]);

        for (**int** j = 0; j < bk.no\_of\_resources; j++)

        {

            scanf(" %d", &bk.max\_req[i][j]);

        }

    }

    printf("\nEnter Allocated instances:\n");

    for (**int** i = 0; i < bk.no\_of\_process; i++)

    {

        printf("Allocated instances for process %s:", bk.process\_name[i]);

        for (**int** j = 0; j < bk.no\_of\_resources; j++)

        {

            scanf(" %d", &bk.allocation[i][j]);

        }

    }

    for (**int** i = 0; i < bk.no\_of\_process; i++)

    {

        bk.f[i] = 0;

        for (**int** j = 0; j < bk.no\_of\_resources; j++)

        {

            bk.need[i][j] = bk.max\_req[i][j] - bk.allocation[i][j];

        }

    }

    return bk;

}

**int** main(**int** argc, **char** **const** \*argv**[]**)

{

    bankersdata bk = getdata();

**int** choice = 0;

    printf("\nMenu:\n\t1.Enter new data\n\t2.PrintData\n\t3.Bankers State\n\t4.Resource Request\n\t5.Exit\nEnter Choice:");

    scanf(" %d", &choice);

    while (choice)

    {

        switch (choice)

        {

        case 1:

        {

            bk = getdata();

            break;

        }

        case 2:

        {

            printdata(&bk);

            break;

        }

        case 3:

        {

            bankersalgo(&bk);

            break;

        }

        case 4:

        {

**char** temp\_process\_name[5];

            printf("\nEnter process id for request:");

            scanf(" %s", &temp\_process\_name);

**int** index\_of\_process = -1;

            for (**int** i = 0; i < bk.no\_of\_process; i++)

            {

                if (strcmp(temp\_process\_name, bk.process\_name[i]) == 0)

                {

                    index\_of\_process = i;

                    break;

                }

            }

            if (index\_of\_process == -1)

            {

                printf("\nprocess name not correct!!!\n");

                break;

            }

            else

            {

**int** allocation\_vector[max];

                printf("\nEnter the request vector for %s:", bk.process\_name[index\_of\_process]);

                for (**int** i = 0; i < bk.no\_of\_resources; i++)

                {

                    scanf(" %d", &allocation\_vector[i]);

                }

**int** flag = 1;

                for (**int** i = 0; i < bk.no\_of\_resources; i++)

                {

                    if (allocation\_vector[i] > bk.available\_instance[i])

                    {

                        flag = 0;

                        break;

                    }

                }

                if (flag == 0)

                {

                    printf("\n!!!Resource cannot be allocated!!!");

                    break;

                }

                for (**int** i = 0; i < bk.no\_of\_resources; i++)

                {

                    bk.available\_instance[i] -= allocation\_vector[i];

                    bk.allocation[index\_of\_process][i] += allocation\_vector[i];

                }

                bankersalgo(&bk);

            }

            break;

        }

        case 5:

            return 0;

        default:

        {

            printf("\n!!!Enter correct choice!!!\n");

            break;

        }

        }

        printf("\nMenu:\n\t1.Enter new data\n\t2.PrintData\n\t3.Bankers State\n\t4.Resource Request\n\t5.Exit\nEnter Choice:");

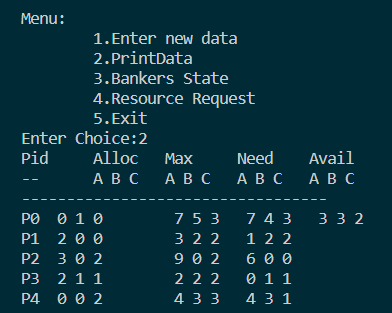
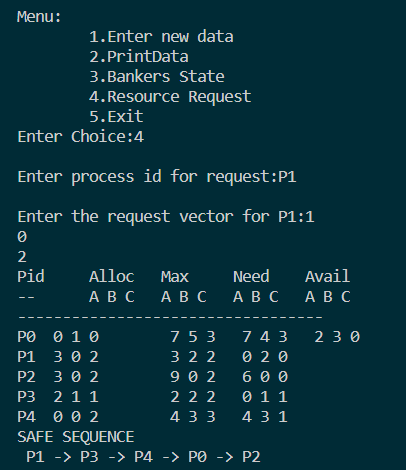
        scanf(" %d", &choice);

    }

    return 0;

}

**Output:**

** **

2. Assignment 2: Develop a C program to implement algorithm for deadlock detection with multiple instances of resource types and display the processes involved in deadlock

**Algorithms**:

1. Get no of process, no of resources, available instances, maximum required matrix, allotted instances from user
2. Calculate need matrix by subtracting allocation matrix from maximum required matrix
3. Set ind to 0
4. For i from 0 to no of process times
   1. If f[i] is equal to 0
      1. Set flag = 0
   2. For j no of resources times
      1. If need[I,j]greater than available[j] then set flag = 1
   3. If flag equal to 0 then
      1. Set safeseq[ind]=i
      2. Increment ind
      3. Set available [j] equal to sum of available[j] and allocstion[I,j]
      4. Set f[i] is equal to 1
5. Set flag equal to 1
6. For i from 0 to no of process
   1. If f[i] is equal to 0 then print not a safe sequence and exit
   2. If flag is equal to 1 then print safe sequence

**Code:**

#include <stdio.h>

*//#include <conio.h>*

#include <string.h>

#define max 100

typedef **struct** bankersdata

{

**int** no\_of\_process;

**int** no\_of\_resources;

**char** process\_name[max][5];

**char** resources\_name[max][5];

**int** available\_instance[max];   *// available\_instance[no\_of\_resources]*

**int** max\_req[max][max];         *// max\_req[no\_of\_process][no\_of\_resources];*

**int** allocation[max][max];      *// allocation[no\_of\_process][no\_of\_resources];*

**int** f[max], safesequence[max]; *// f[no\_of\_process],safesequence[no\_of\_process]*

**int** need[max][max];

} bankersdata;

**void** printdata(bankersdata \*bk)

{

    printf("Pid\tAlloc\tMax  \tNeed \tAvail\t\n--\t");

    for (**int** i = 0; i < bk->no\_of\_resources; i++)

    {

        printf("%s ", bk->resources\_name[i]);

    }

    printf("\t");

    for (**int** i = 0; i < bk->no\_of\_resources; i++)

    {

        printf("%s ", bk->resources\_name[i]);

    }

    printf("\t");

    for (**int** i = 0; i < bk->no\_of\_resources; i++)

    {

        printf("%s ", bk->resources\_name[i]);

    }

    printf("\t");

    for (**int** i = 0; i < bk->no\_of\_resources; i++)

    {

        printf("%s ", bk->resources\_name[i]);

    }

    printf("\t");

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

    for (**int** i = 0; i < bk->no\_of\_process; i++)

    {

        printf("%s ", bk->process\_name[i]);

        for (**int** j = 0; j < bk->no\_of\_resources; j++)

        {

            printf(" %d", bk->allocation[i][j]);

        }

        printf("\t");

        for (**int** j = 0; j < bk->no\_of\_resources; j++)

        {

            printf(" %d", bk->max\_req[i][j]);

        }

        printf("\t");

        for (**int** j = 0; j < bk->no\_of\_resources; j++)

        {

            printf(" %d", bk->need[i][j]);

        }

        printf("\t");

        if (i == 0)

        {

            for (**int** j = 0; j < bk->no\_of\_resources; j++)

            {

                printf(" %d", bk->available\_instance[j]);

            }

        }

        printf("\t\n");

    }

}

**void** bankersalgo(bankersdata \*bk)

{

**int** ind = 0;

**int** y = 0;

    for (**int** k = 0; k < 5; k++)

    {

        for (**int** i = 0; i < bk->no\_of\_process; i++)

        {

            if (bk->f[i] == 0)

            {

**int** flag = 0;

                for (**int** j = 0; j < bk->no\_of\_resources; j++)

                {

                    if (bk->need[i][j] > bk->available\_instance[j])

                    {

                        flag = 1;

                        break;

                    }

                }

                if (flag == 0)

                {

                    bk->safesequence[ind] = i;

                    ind++;

                    for (**int** j = 0; j < bk->no\_of\_resources; j++)

                    {

                        bk->available\_instance[j] += bk->allocation[i][j];

                    }

                    bk->f[i] = 1;

                }

            }

        }

    }

**int** flag = 1;

    for (**int** i = 0; i < bk->no\_of\_process; i++)

    {

        if (bk->f[i] == 0)

        {

            flag = 0;

            printf("\n\n!!!!NOT A SAFE SYSTEM!!!!\nDue to the following processes:");

            break;

        }

    }

    if (flag == 1)

    {

        printf("SAFE SEQUENCE\n");

        for (**int** i = 0; i < bk->no\_of\_process - 1; i++)

        {

            printf(" %s ->", bk->process\_name[bk->safesequence[i]]);

        }

        printf(" %s", bk->process\_name[bk->safesequence[bk->no\_of\_process - 1]]);

    }

    else

    {

        for (**int** i = 0; i < bk->no\_of\_process; i++)

        {

            if (bk->f[i] == 0)

            {

                printf(" %s ", bk->process\_name[i]);

            }

        }

    }

}

bankersdata getdata()

{

    bankersdata bk;

    printf("\nEnter no of process:");

    scanf("%d", &bk.no\_of\_process);

    printf("\nEnter process ids:\n");

    for (**int** i = 0; i < bk.no\_of\_process; i++)

    {

        printf("process name of process %d:", i + 1);

        scanf(" %s", &bk.process\_name[i]);

    }

    printf("\nEnter no of resources:");

    scanf("%d", &bk.no\_of\_resources);

    printf("\nEnter resource ids:\n");

    for (**int** i = 0; i < bk.no\_of\_resources; i++)

    {

        printf("resource name of resource %d:", i + 1);

        scanf(" %s", &bk.resources\_name[i]);

    }

    printf("\nEnter available instances:\n");

    for (**int** i = 0; i < bk.no\_of\_resources; i++)

    {

        printf("available instances of resource %s:", bk.resources\_name[i]);

        scanf(" %d", &bk.available\_instance[i]);

    }

    printf("\nEnter Maximum requirement:\n");

    for (**int** i = 0; i < bk.no\_of\_process; i++)

    {

        printf("Maximum requirement for process %s:", bk.process\_name[i]);

        for (**int** j = 0; j < bk.no\_of\_resources; j++)

        {

            scanf(" %d", &bk.max\_req[i][j]);

        }

    }

    printf("\nEnter Allocated instances:\n");

    for (**int** i = 0; i < bk.no\_of\_process; i++)

    {

        printf("Allocated instances for process %s:", bk.process\_name[i]);

        for (**int** j = 0; j < bk.no\_of\_resources; j++)

        {

            scanf(" %d", &bk.allocation[i][j]);

        }

    }

    for (**int** i = 0; i < bk.no\_of\_process; i++)

    {

        bk.f[i] = 0;

        for (**int** j = 0; j < bk.no\_of\_resources; j++)

        {

            bk.need[i][j] = bk.max\_req[i][j] - bk.allocation[i][j];

        }

    }

    return bk;

}

**int** main(**int** argc, **char** **const** \*argv**[]**)

{

    bankersdata bk = getdata();

**int** choice = 0;

    printf("\nMenu:\n\t1.Enter new data\n\t2.PrintData\n\t3.Bankers State\n\t4.Exit\nEnter Choice:");

    scanf(" %d", &choice);

    while (choice)

    {

        switch (choice)

        {

        case 1:

        {

            bk = getdata();

            break;

        }

        case 2:

        {

            printdata(&bk);

            break;

        }

        case 3:

        {

            bankersalgo(&bk);

            break;

        }

        case 4:

        {

            return 0;

            break;

        }

        default:

        {

            printf("\n!!!Enter correct choice!!!\n");

            break;

        }

        }

        printf("\nMenu:\n\t1.Enter new data\n\t2.PrintData\n\t3.Bankers State\n\t4.Exit\nEnter Choice:");

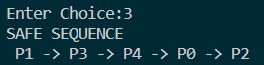
        scanf(" %d", &choice);

    }

    return 0;

}

**Output:**

****

**Learning Outcome:**

* Bankers algorithm implementation
* Importance of deadlock prevention
* Printing safe Sequence