Ex no: 7 M.Rohith

Date: 15-05-2023 3122 21 5001 085

SSN COLLEGE OF ENGINEERING, KALAVAKKAM
(An Autonomous Institution, Affiliated to Anna University, Chennai)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
II Year CSE – A & B Sections (IV Semester)

Academic Year 2022-23

UCS2412 - OPERATING SYSTEMS LAB

Lab exercise 7 Implementation of Banker's algorithm (deadlock avoidance)

Aim:
Develop a C program to implement the Banker's algorithm for deadlock avoidance

Algorithm:

1. Read the following
    a. Number of processes.
    b. Number of resources and number of instances of each resource available.
    c. Maximum requirement of each process,
    d. Allocated instances of resources
2. Determine the need of each process
3. Repeat the following till all processes are done.
    a. Check if request of process i less than or equal to need of that process
        i. If yes proceed
        ii. Otherwise raise an error condition
    b. Check if request of process i less than or equal to available instances
        i. If yes proceed
        ii. Otherwise wait till available.
    c. Update the available vector, allocation vector and need vector
    d. Generate safety sequence by running safety algorithm.

Sample Input/Output:

Banker's Algorithm
1. Read Data
2. Print Data
3. Safety Sequence
4. Exit

Enter the option :1

Number of processes: 5 P0, P1, P2, P3, P4
Number of resources: 3 A B C

Ex no: 7 M.Rohith

Date: 15-05-2023 3122 21 5001 085

Number of Available instances of A: 3
Number of Available instances of B: 3
Number of Available instances of C: 2

Maximum requirement for P0: 7 5 3
Maximum requirement for P1: 3 2 2
Maximum requirement for P2: 9 0 2
Maximum requirement for P3: 2 2 2
Maximum requirement for P4: 4 3 3

Allocated instances to P0: 0 1 0
Allocated instances to P1: 2 0 0
Allocated instances to P2: 3 0 2
Allocated instances to P3: 2 1 1
Allocated instances to P4: 0 0 2

Enter the option: 2

Pid	Alloc	Max	Need	Avail
	A B C	A B C		A B C A B C
P0	0 1 0	7 5 3	* * *	3 3 2
P1	2 0 0	3 2 2	* * *	
P2	3 0 2	9 0 2	* * *	
P3	2 1 1	2 2 2	* * *	
P4	0 0 2	4 3 3	* * *	

Enter the option: 3

Display the Safety Sequence:
* * * * *

Enter the option:4

Ex no: 7 M.Rohith

Date: 15-05-2023 3122 21 5001 085

Program code:

```
#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);
```

Ex no: 7 M.Rohith

Date: 15-05-2023 3122 21 5001 085

```
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 - 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:
            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)
{
    Process p[MAX * 2];
    Resource avail_copy[MAX];
    for (int i = 0; i < n_process; i++)
        p[i] = arr[i];
```

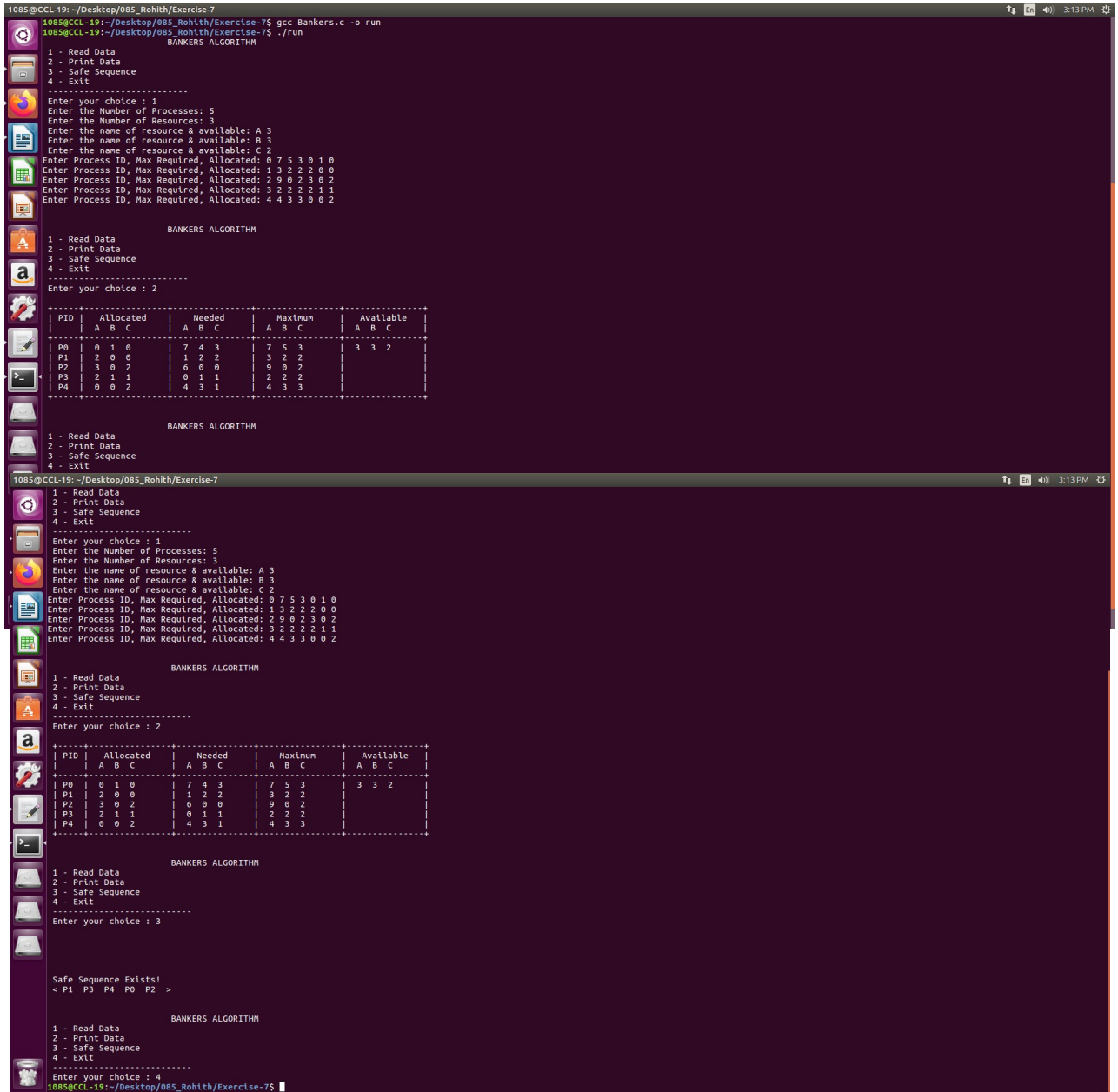
```
  for (int i = 0; i < n_resources; i++)
    avail_copy[i] = avail[i];

  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");
  }

  if (completed == n_process)
  {
    printf(" Safe Sequence Exists!\n");
```

Ex no: 7 M.Rohith

Date: 15-05-2023 3122 21 5001 085

```
        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;
}
```

Output:



Ex no: 7 M.Rohith

Date: 15-05-2023 3122 21 5001 085

Learning Outcomes:

Thus Bankers algorithm for the given problem has been implemented and the safe sequence has been obtained.