Department of Computer Science and Engineering

S.G.Shivanirudh , 185001146, Semester IV

19 March 2020

UCS1411 - Operating Systems Laboratory

Lab Exercise 7: Implementation of Banker's Algorithm (Deadlock Avoidance)

Objective:

Develop a C program to implement the banker's algorithm for deadlock avoidance.

Code:

Q.To write a C program to implement the banker's algorithm for deadlock avoidance.

```

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

```

```c
struct Job{
    char *PID;
    int maxReq[100];
    int alloc[100];
    int need[100];
};

typedef struct Job Process;

void initialise(Process *p,int NoR){
    p->PID=(char*)malloc(100);
    for(int i =0;i<NoR;i++){
        p->maxReq[i]=0;
        p->alloc[i]=0;
        p->need[i]=0;
    }
}

void acceptProcess(Process *p,int NoR){

    printf("\nEnter PID: ");scanf(" %s",p->PID);
    printf("Enter Maximum vector for the process: ");
    for(int i=0;i<NoR;i++)
        scanf("%d",&p->maxReq[i]);

    printf("Enter Allocation vector for the process: ");
    for(int i=0;i<NoR;i++)
        scanf("%d",&p->alloc[i]);

    for(int i=0;i<NoR;i++)
        p->need[i]=p->maxReq[i]-p->alloc[i];
}

int vectorCheck(int vec1[],int vec2[],int NoR){
    for(int i=0;i<NoR;i++)
        if(vec1[i]>vec2[i])
            return 0;
    return 1;
}

int checkFlag(int complete_flag[],int NoP){
    for(int i=0;i<NoP;i++)
        if(complete_flag[i]==0)
            return 1;
    return 0;
```

```c
}


int safetySequence(Process p[],int NoP,int NoR,int *available){
    //Safe sequence
    char *safeSeq[100];
    for(int i=0;i<100;i++)
        safeSeq[i]=(char*)malloc(100);

    int ssctr=0,tmpssctr=0;

    //List of complete processes
    int complete_flag[100];

    //Initialise all processes as incomplete
    for(int i=0;i<NoP;i++)
        complete_flag[i]=0;
    for(int pno=0;pno<NoP;pno++){

        if(pno==0)
            tmpssctr=ssctr;
        //Check if an incomplete process has its need less than the available resources
        if(vectorCheck(p[pno].need,available,NoR)&&!complete_flag[pno]){

            strcpy(safeSeq[ssctr++],p[pno].PID);

            //Set process as complete
            complete_flag[pno]=1;

            //Add allocated resources to the available resources
            for(int i=0;i<NoR;i++)
                available[i]+=p[pno].alloc[i];
        }
        if(pno==NoP-1){
            //Check if all processes are complete
            if (checkFlag(complete_flag,NoP))
                pno=-1;
            //If no incomplete process that has need less than available can be found, break the loop
            if(tmpssctr==ssctr)
                break;
        }
    }
    if(tmpssctr==ssctr){
        printf("\n The system is not in a safe state. \n");
        return 0;
    }
    else{
        printf("\n The safe sequence is: \n <");
        for(int i=0;i<ssctr;i++){
            printf(" %s ",safeSeq[i]);
        }
        printf(">\n");
        printf("\n The system is in a safe state. \n");
        return 1;
    }
}

void RequestCheck(Process p[],int NoP,int NoR,char* reqProcess,int request[],int available[]){
    int pno=0;
    for(;pno<NoP;pno++){
        if(strcmp(p[pno].PID,reqProcess)==0)
            break;
    }

    //Check if the request made is less than the maximum resources that the process requires
    if(vectorCheck(request,p[pno].need,NoR)){

        //Check if the request made is less than the available resources
        if(vectorCheck(request,available,NoR)){

            //Pretend to satisfy the request
            for(int i=0;i<NoR;i++){
                p[pno].alloc[i]+=request[i];
                p[pno].need[i]-=request[i];
                available[i]-=request[i];
            }
            //Check if the system is in a safe state
            if(safetySequence(p,NoP,NoR,available))
                printf("\n %s's request can be granted immediately. \n",reqProcess);
            else
                printf("\n %s's request cannot be granted
   immediately. Process has to wait. \n",reqProcess);
        }
        else{
            printf("\n Available resources not enough to
   satisfy request. Process has to wait. \n");
        }
    }
    else{
        printf("\nRequest exceeding maximum specified need of
    the process. \n");
    }
}

int main(){
    int NoP; //Number of processes
    int NoR; //Number of resources

    int maxInstance[100]; //Maximum instances of each
   resource

    int available[100]; //Available number of resources in
   the system

    Process p[100];     //Process list

    int option;
    do{

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

        printf("\n 1.Read data \n 2.Print data \n 3.Safety
   Sequence \n 4.Request \n 0.Exit \n");
        printf("\n Enter option: ");scanf("%d",&option);

        if(option==1){
            printf("\n Enter number of processes: ");scanf("%
   d",&NoP);

            printf("\n Enter number of resources: ");scanf("%
   d",&NoR);
```

```

            for(int i=0;i<NoR;i++){
                printf("\n Enter maximum instance of resource %c : ",'A'+i);scanf("%d",&maxInstance[i]);
            }

            for(int i=0;i<NoP;i++){
                initialise(&p[i],NoR);
                acceptProcess(&p[i],NoR);
            }

            for(int i=0;i<NoR;i++){
                int sum=0;
                for(int j=0;j<NoP;j++)
                    sum+=p[j].alloc[i];
                available[i]=maxInstance[i]-sum;
            }
        }
        else if(option==2){
            printf("%13s %13s %13s %13s\n","Max","Alloc","Need","Available");
            for(int i=0;i<4;i++){
                printf("%5s"," ");
                for(int j=0;j<NoR;j++){
                    printf("%2c",'A'+j);
                }
                for(int j=0;j<=8-NoR*2;j++)
                    printf(" ");
            }
            printf("\n");
            for(int i=0;i<NoP;i++){
                printf("%3s  ",p[i].PID);
                for(int j=0;j<NoR;j++){
                    printf("%2d",p[i].maxReq[j]);
                }
                for(int j=0;j<=8-NoR*2;j++)
                    printf(" ");

                printf("%5s"," ");
                for(int j=0;j<NoR;j++){
                    printf("%2d",p[i].alloc[j]);
                }
                for(int j=0;j<=8-NoR*2;j++)
                    printf(" ");

                printf("%5s"," ");
                for(int j=0;j<NoR;j++){
                    printf("%2d",p[i].need[j]);
                }
                for(int j=0;j<=8-NoR*2;j++)
                    printf(" ");

                if(i==0){
                    printf("%5s"," ");
                    for(int j=0;j<NoR;j++){
                        printf("%2d",available[j]);
                    }
                    for(int j=0;j<=8-NoR*2;j++)
                        printf(" ");
                }
                printf("\n");
            }
        }
        else if(option==3){
            safetySequence(p,NoP,NoR,available);
        }
        else if(option==4){
            for(int i=0;i<NoR;i++){
                int sum=0;
                for(int j=0;j<NoP;j++)
                    sum+=p[j].alloc[i];
                available[i]=maxInstance[i]-sum;
            }

            char* reqProcess=(char*)malloc(100);

            int request[100];

            printf("\nEnter PID of process making request: ");scanf(" %s",reqProcess);
            printf("\nEnter request vector: ");
            for(int i=0;i<NoR;i++)
                scanf("%d",&request[i]);

            RequestCheck(p,NoP,NoR,reqProcess,request,available);
        }
        else if(option!=0){
            printf("\n Invalid input \n");
        }
        else;
    }while(option);
}
```

Output:

```

 ---------------------------------------------------

                 BANKERS ALGORITHM

 ---------------------------------------------------


 1.Read data
 2.Print data
 3.Safety Sequence
 4.Request
 0.Exit

 Enter option: 1

 Enter number of processes: 5

 Enter number of resources: 4

 Enter maximum instance of resource A : 3

 Enter maximum instance of resource B : 14

 Enter maximum instance of resource C : 12

 Enter maximum instance of resource D : 12

Enter PID: P0
Enter Maximum vector for the process: 0 0 1 2
Enter Allocation vector for the process: 0 0 1 2

Enter PID: P1
Enter Maximum vector for the process: 1 7 5 0
Enter Allocation vector for the process: 1 0 0 0

Enter PID: P2
Enter Maximum vector for the process: 2 3 5 6
Enter Allocation vector for the process: 1 3 5 4

Enter PID: P3
Enter Maximum vector for the process: 0 6 5 2
Enter Allocation vector for the process: 0 6 3 2

Enter PID: P4
Enter Maximum vector for the process: 0 6 5 6
Enter Allocation vector for the process: 0 0 1 4

 ---------------------------------------------------

                 BANKERS ALGORITHM

 ---------------------------------------------------


 1.Read data
 2.Print data
 3.Safety Sequence
 4.Request
 0.Exit

 Enter option: 2
          Max         Alloc          Need     Available
      A B C D       A B C D       A B C D       A B C D
 P0   0 0 1 2       0 0 1 2       0 0 0 0       1 5 2 0
 P1   1 7 5 0       1 0 0 0       0 7 5 0
 P2   2 3 5 6       1 3 5 4       1 0 0 2
 P3   0 6 5 2       0 6 3 2       0 0 2 0
 P4   0 6 5 6       0 0 1 4       0 6 4 2

 ---------------------------------------------------

                 BANKERS ALGORITHM

 ---------------------------------------------------


 1.Read data
 2.Print data
 3.Safety Sequence
 4.Request
 0.Exit

 Enter option: 3

 The safe sequence is:
 < P0  P2  P3  P4  P1 >

 The system is in a safe state.

 ---------------------------------------------------

                 BANKERS ALGORITHM

 ---------------------------------------------------


 1.Read data
 2.Print data
 3.Safety Sequence
 4.Request
 0.Exit

 Enter option: 4

Enter PID of process making request: P1

Enter request vector: 0 4 2 0

 The safe sequence is:
 < P0  P2  P3  P4  P1 >

 The system is in a safe state.

 P1's request can be granted immediately.

 ---------------------------------------------------

                 BANKERS ALGORITHM

 ---------------------------------------------------


 1.Read data
 2.Print data
 3.Safety Sequence
 4.Request
 0.Exit

```

```
Enter option: 0
```