IMPLEMENTATION

```
/*
Roll No         : B21CSB69
Name            : Sreelal V
Experiment No : 4.1
*/

#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<time.h>
#include<semaphore.h>
#include<pthread.h>

#define BUFF_SIZE 10

int buffer[BUFF_SIZE] ;
int count = -1 ;
int in = 0 ;
int out = 0 ;

sem_t semaphore ;


void* producer(void * arg){
   while(1){
      int y = rand() % 100 ;
      sem_wait(&semaphore) ;
      if ( count < BUFF_SIZE ){
         buffer[++count] = y ;
         printf("produced = %d\n",y) ;
      }
      sem_post(&semaphore) ;
      sleep(1);
   }
   return NULL ;
}

void* consumer(void * arg){
   while(1){
      int y ;
      sem_wait(&semaphore) ;
      if ( count >= 0 ){
         y = buffer[count--] ;
         printf("Consumed : %d\n",y) ;
      }
      sem_post(&semaphore) ;
      sleep(1) ;
   }
}
```

```
int main(){
    pthread_t p, c ;
    srand(time(NULL)) ;

    sem_init(&semaphore, 0, 1);


    pthread_create(&p, NULL , producer, NULL) ;
    pthread_create(&c, NULL , consumer, NULL) ;

    pthread_join(p, NULL) ;
    pthread_join(c, NULL) ;

    sem_destroy(&semaphore) ;


}
```

output:

```
produced = 12
Consumed : 12
produced = 74
Consumed : 74
produced = 39
Consumed : 39
produced = 89
produced = 76
Consumed : 76
produced = 85
Consumed : 85
produced = 72
Consumed : 72
Consumed : 89
produced = 45
produced = 34
Consumed : 34
produced = 90
Consumed : 90
produced = 73
Consumed : 73
Consumed : 45
produced = 22
Consumed : 22
produced = 23
Consumed : 23
produced = 74
Consumed : 74
produced = 68
Consumed : 68
produced = 30
Consumed : 30
```

IMPLEMENTATION

```
/*
Roll No         : B21CSB69
Name            : Sreelal V
Experiment No : 4.2
*/
#include<stdio.h>
#include<unistd.h>
#include<pthread.h>
#include<semaphore.h>

#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (noPhil + 4) % N
#define RIGHT (noPhil + 1) % N

int state[N];
int phil[N] = { 0, 1, 2, 3, 4 };
sem_t mutex;
sem_t S[N];

void test(int noPhil){
   if (state[noPhil] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING) {

      state[noPhil] = EATING;
      sleep(2);
      printf("Philosopher %d takes fork %d and %d\n", noPhil + 1, LEFT + 1, noPhil + 1);
      printf("Philosopher %d is Eating\n", noPhil + 1);
      sem_post(&S[noPhil]);
   }
}

void take_fork(int noPhil){
   sem_wait(&mutex);

   state[noPhil] = HUNGRY;
   printf("Philosopher %d is Hungry\n", noPhil + 1);

   test(noPhil);
   sem_post(&mutex);

   sem_wait(&S[noPhil]);
   sleep(1);
}

void put_fork(int noPhil){
   sem_wait(&mutex);

   state[noPhil] = THINKING;
```

```
    printf("Philosopher %d putting fork %d and %d down\n",noPhil + 1,
    LEFT + 1, noPhil + 1);
    printf("Philosopher %d is thinking\n", noPhil + 1);
    test(LEFT);
    test(RIGHT);
    sem_post(&mutex);
}

void* philosopher(void* num){
    while (1) {
        int* i = num;
        sleep(1);
        take_fork(*i);
        sleep(0);
        put_fork(*i);
    }
}

void main(){
    int i;
    pthread_t thread_id[N];

    sem_init(&mutex, 0, 1);
    for (i = 0; i < N; i++)
    sem_init(&S[i], 0, 0);
    for (i = 0; i < N; i++) {

        pthread_create(&thread_id[i], NULL,philosopher, &phil[i]);
        printf("Philosopher %d is thinking\n", i + 1);
    }
    for (i = 0; i < N; i++){
    pthread_join(thread_id[i], NULL);
}
}
```

output:

```
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 3 is Hungry
Philosopher 2 is Hungry
Philosopher 1 is Hungry
Philosopher 5 is Hungry
Philosopher 4 is Hungry
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 3 takes fork 2 and 3
```

Philosopher 3 is Eating
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 4 is Hungry
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 3 is Hungry
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 5 is Hungry
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 2 is Hungry
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 4 is Hungry
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 1 is Hungry
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 3 is Hungry
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking

IMPLEMENTATION

```
/*
Roll No         : B21CSB69
Name            : Sreelal V
Experiment No : 4.3
*/

#include <stdio.h>

int Max[10][10], need[10][10], alloc[10][10],avail[10],tot[10],work[10], finish[10],
safeSequence[10];
int p, r, i, j, process, count=0;

void safesequence(){
   printf("\n Max matrix:\tAllocation matrix:\n");
   for(i = 0; i < p; i++){
      for( j = 0; j < r; j++)
         printf("%d ", Max[i][j]);
      printf("\t\t");
      for( j = 0; j < r; j++)
         printf("%d ", alloc[i][j]);
      printf("\n");
   }
   do{
      process = -1;
      for(i = 0; i < p; i++){
         if(finish[i] == 0){
            process = i ;
            for(j = 0; j < r; j++){
               if(work[j] < need[i][j]){
                  process = -1;
                  break;
               }
            }
         }
         if(process != -1)
            break;
      }
      if(process != -1){
         safeSequence[count] = process ;
         count++;
         for(j = 0; j < r; j++){
            work[j] += alloc[process][j];
         }
         finish[process] = 1;
      }
   }while(count != p && process != -1);
   if(count == p){
      printf("\nThe system is in a safe state\n");
      printf("Safe Sequence : ");
      for( i = 0; i < p; i++)
```

```
            printf("P%d ", safeSequence[i]);
        printf("\n");
    }else
        printf("\nThe system is in an unsafe state\n");
}

void resourceAlloc(){
    int resource[10][10];
    int granted=1;
    printf("Enter the resource allocation : \n");
    for(i = 0; i < p; i++) {
        printf("\nFor process %d : ",i );
        for(j = 0; j < r; j++)
        scanf("%d", &resource[i][j]);
    }
    for(int i=0;i<p;i++){
        for(int j=0;j<r;j++){
            if(resource[i][j] > need[i][j]){
                granted=0;
                break;
            }
            if(resource[i][j] > avail[j]){
                granted=0;
                break;
            }
        }
        if(granted==1){
            avail[j] -= resource[i][j];
            alloc[i][j] += resource[i][j];
            need[i][j] -= resource[i][j];
        }
    }
    if(granted==1)
        printf("Resource request granted\n");
    safesequence();
}

void main(){
    printf("Enter the no of processes : ");
    scanf("%d", &p);
    for(i = 0; i< p; i++)
        finish[i] = 0;
    printf("\nEnter the no of resources : ");
    scanf("%d", &r);
    printf("\nEnter the total resources available : ");
    for(int i = 0;i< r;i++){
        scanf("%d",&tot[i]);
    }
    printf("\n\nEnter the Max Matrix for each process : ");
    for(i = 0; i < p; i++){
        printf("\nFor process %d : ", i );
        for(j = 0; j < r; j++)
```

```
            scanf("%d", &Max[i][j]);
    }
    printf("\n\nEnter the allocation for each process : ");
    for(i = 0; i < p; i++){
        printf("\nFor process %d : ",i );
        for(j = 0; j < r; j++)
            scanf("%d", &alloc[i][j]);
    }
    for(int i =0;i<r;i++){
        int allo=0;
        for(int j =0;j<p;j++){
            allo+=alloc[j][i];
        }
        avail[i]=tot[i]-allo;
    }
    printf("\nAvailable Resources : ");
    for(int i = 0; i < r; i++)  {
        printf("%d",avail[i]);
        work[i]=avail[i];
        finish[i]=0;
    }
    for(i = 0; i < p; i++)
        for(j = 0; j < r; j++)
            need[i][j] = Max[i][j] - alloc[i][j];
    safesequence();
    for(int i =0;i<r;i++){
        work[i]=avail[i];
        finish[i]=0;
    }
    resourceAlloc();
}
```

output:

Enter the no of processes : 3

Enter the no of resources : 3

Enter the total resources available : 8 5 4

Enter the Max Matrix for each process :
For process 0 : 6 2 0

For process 1 : 3 3 3

For process 2 : 8 4 3

Enter the allocation for each process :
For process 0 : 3 2 0

For process 1 : 2 1 1

For process 2 : 0 0 1

Available Resources : 322
Max matrix: Allocation matrix:
6 2 0 3 2 0
3 3 3 2 1 1
8 4 3 0 0 1

The system is in a safe state
Safe Sequence : P0 P1 P2
Enter the resource allocation :

For process 0 : 0 0 0

For process 1 : 0 0 0

For process 2 : 0 0 2
Resource request granted

Max matrix: Allocation matrix:
6 2 0 3 2 0
3 3 3 2 1 1
8 4 3 0 0 1

The system is in an unsafe state