view source
print?

```c
//Bankers Algorithm
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>

#define NUMBER_OF_CUSTOMERS 5             /* maximum number of processes                   */
#define NUMBER_OF_RESOURCES 3              /* maximum number of resource types              */
int ProcCurr[5][3];     /* 3 threads(processes), 3 resources    */
int temp[5][3];       /* temp array location           */
int Available[NUMBER_OF_RESOURCES];        /* Available[m] = # resources unallocated */
int Max[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES]; /* Max[n][m] = max demand of processes n for resource m    */
int Allocation[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES] = { {1,2,3},{3,2,1},{1,1,1},{1,1,1},{1,1,1} }; /* Allocation[n][m] = # resources m allocated to processes n*/
int Need[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES];       /* Need[n][m] = resources m needed by processes n         */
int counti = 0;             /* Need[n][m] = Max[n][m] - Allocation[n][m]     */
int countj = 0;
int threadsi = 5;
int threadsj = 3;

void *inc_count(void *r);
void *watch_count(void *r);

pthread_mutex_t mutex; /*mutex id*/
pthread_cond_t count_threshold_cv;

int main(){

  long r1 = 1,r2 = 2,r3 = 3;
srand(time(NULL));
int x, y;
    for(x=0; x<NUMBER_OF_CUSTOMERS; x++){
      for(y=0; y<NUMBER_OF_RESOURCES; y++){
        if(y==0){
          Max[x][y] = rand() % r1 + 1;
        }
      else if(y==1){
          Max[x][y] = rand() % r2 + 1;
        }
      else{
          Max[x][y] = rand() % r3 + 1;
        }

      }
    }


pthread_t ProcCurr[5][3]; /*id of thread*/
pthread_attr_t attr;
int  i, j;


  printf("Enter Resource 1: ");     /* write a prompt */
  scanf("%ld", &r1);

  printf("Enter Resource 2: ");     /* write a prompt */
  scanf("%ld", &r2);

  printf("Enter Resource 3: ");     /* write a prompt */
  scanf("%ld", &r3);



if(pthread_mutex_init(&mutex, NULL) < 0){
   perror("Pthread_mutex_init error.");
   exit(1);
   }
else
   //pthread_mutex_init(&mutex, NULL);

pthread_cond_init(&count_threshold_cv, NULL);

pthread_attr_init(&attr); /*get default attributes*/

pthread_create(&ProcCurr[0][0], &attr, watch_count, (void *)r1);
pthread_create(&ProcCurr[1][0], &attr, inc_count, (void *)r2);
```

```
  pthread_create(&ProcCurr[2][0], &attr, inc_count, (void *)r3);


  for(i=0; i<=threadsi; i++){
      for(j=0; j<=threadsj; j++){
          pthread_join(ProcCurr[i][j],NULL); /*wait for thread to exit*/
          }
      }
  printf("Main: waited on %d, %d threads. Done.\n", threadsi, threadsj);

  pthread_attr_destroy(&attr);
  pthread_mutex_destroy(&mutex);
  pthread_cond_destroy(&count_threshold_cv);
  pthread_exit(NULL);

}

void *inc_count(void *r)
{  /*processes are running, thread of process is initalize to something <=3, each threads request up to 3 resources, when all resources are commited then next thread will have to wait (mutex goes to resource from a thread letting other threads know not to this resource)*/
  int i, j, n, m;
  long my_id = (long)r;

  for(i=0; i<10; i++){
      for(j=0; j<10; j++){
    Need[n][m] = Max[n][m] - Allocation[i][j];
    printf("Allocation = %d, Need = %d\n", Allocation[i][j], Need[n][m]);
    }
    pthread_mutex_lock(&mutex);
    if(counti == NUMBER_OF_CUSTOMERS && countj == NUMBER_OF_RESOURCES){
       pthread_cond_signal(&count_threshold_cv);
       printf("inc_count: thread %ld, Need = %d. Threshold reached.\n",my_id, Need[n][m]);
       }
    printf("inc_count: thread %ld, Need = %d. Unlocking mutex.\n", my_id, Need[n][m]);
    pthread_mutex_unlock(&mutex);
    sleep(1);
    watch_count(r);
    }
  pthread_exit(NULL);
  watch_count(r);

```

```
}

void *watch_count(void *r)
{
  long my_id = (long)r;
  int n, m;

  printf("Start watch_count: thread %ld\n", my_id);

   while(counti < NUMBER_OF_CUSTOMERS && countj <NUMBER_OF_RESOURCES)
  { pthread_mutex_lock(&mutex);
   Available[n] = Max[n][m] - Allocation[counti++][countj++];
   printf("Available = %d\n", Available[n]);
   pthread_cond_wait(&count_threshold_cv, &mutex);
   printf("watch_count: thread %ld, available = %d. Conditional Signal
Received.\n", my_id, Available[m]);
   countj++;
   printf("watch_count: thread %ld, Need now = %d.\n", my_id, Need[counti]
[countj]);
  }
  pthread_mutex_unlock(&mutex);
  pthread_exit(NULL);
}
```