Student Name: AYUSH DWIVEDI
Student ID 11801780
Email Address: akdrishu@gmail.com
GitHub Link: https://github.com/akdrishu/OS_PROJECT
Question-Code: Q3) Write a multithreaded program that implements the banker's algorithm. Create n threads that request and release resources from the bank. The banker will grant the request only if it leaves the system in a safe state. It is important that shared data be safe from concurrent access. To ensure safe access to shared data, you can use mutex locks.

Name: **AYUSH DWIVEDI**

Reg. number: **11801780**

Course Code: **P132-H**

Course Title: **B. Tech. (Computer Science & Engineering) (Hons.)**

SECTION **K18KK**

ROLL NO **31**

Student Name: AYUSH DWIVEDI
Student ID 11801780
Email Address: akdrishu@gmail.com
GitHub Link: https://github.com/akdrishu/OS_PROJECT
Question-Code: Q3) Write a multithreaded program that implements the banker's algorithm. Create n threads that request and release resources from the bank. The banker will grant the request only if it leaves the system in a safe state. It is important that shared data be safe from concurrent access. To ensure safe access to shared data, you can use mutex locks.

CODE : https://github.com/akdrishu/OS_PROJECT/blob/master/bankersalgo.cpp

```cpp
/* So the objective is to avoid deadlock using bankers's algorithm.
 Banker's algorithm states that resource allocation should be done only if the system is in safe state.
 If the system is in unsafe state, there may be chances that there is a deadlock in the system.
 Hence our objective will be to find out if the system is in a safe state or not.
 */
#include <bits/stdc++.h>
#include<conio.h>>
#include <unistd.h>  //for linux
//#include "Windows.h" //for windows

/*if bits/stdc+++.h doesn't work,include all required header files like iostream.h, stdlib.h,time.h,stdio.h,pthread.h,stdbool.h,conio.h*/
using namespace std;
typedef long long int ll;
const ll mxn = 1e2;
ll no_Resources, no_Processes;
ll avail[mxn], allocated[mxn][mxn], no_completedprocess = 0;
ll maxRequired[mxn][mxn], need[mxn][mxn], safeSeq[mxn];

pthread_mutex_t lock_Resources;
pthread_cond_t all_condition;

bool isSafe();

void *Code_Processing(void *arg);

void get_input();

void calculate_needmatrix();

void solve();

void Process_Execution();

int main(int argc, char **argv) {

    get_input();
    calculate_needmatrix();
    solve();
    Process_Execution();
    getch();

}
```

```cpp
void get_input() {
    srand(time(NULL));

    printf("\nNumber of processes? ");
    scanf("%lld", &no_Processes);

    printf("\nNumber of resources? ");
    scanf("%lld", &no_Resources);


    printf("\nCurrently Available resources (R1 R2 ...Rn)? ");
    for (ll i = 0; i < no_Resources; i++)
        scanf("%lld", &avail[i]);


    printf("\n");
    for (ll i = 0; i < no_Processes; i++) {
        printf("\nResource allocated to process %lld (R1 R2 ...)? ", i + 1);
        for (ll j = 0; j < no_Resources; j++)
            scanf("%lld", &allocated[i][j]);
    }
    printf("\n");

    // maximum required resources
    for (ll i = 0; i < no_Processes; i++) {
        printf("\nMaximum resource required by process %lld (R1 R2 ...)? ", i + 1);
        for (ll j = 0; j < no_Resources; j++)
            scanf("%lld", &maxRequired[i][j]);
    }
    printf("\n");
}

/*Function to calculate need matrix
 * need[i][j]= maxRequired[i][j] - allocated[i][j]*/

void calculate_needmatrix() {
    // calculate need matrix
    for (ll i = 0; i < no_Processes; i++) {
        for (ll j = 0; j < no_Resources; j++)
            need[i][j] = maxRequired[i][j] - allocated[i][j];
    }
}
```

Student Name: AYUSH DWIVEDI
Student ID 11801780
Email Address: akdrishu@gmail.com
GitHub Link: https://github.com/akdrishu/OS_PROJECT
Question-Code: Q3) Write a multithreaded program that implements the banker's algorithm. Create n threads that request and release resources from the bank. The banker will grant the request only if it leaves the system in a safe state. It is important that shared data be safe from concurrent access. To ensure safe access to shared data, you can use mutex locks.

```
/*Funtion to find out if safe sequence exists or not and further
 * if safe sequence exists, execute all processes one by one.*/

void solve() {
    // get safe sequence

    for (ll i = 0; i < no_Processes; i++) safeSeq[i] = -1;

    if (!isSafe()) {
        printf("\nNo safe sequence detected, hence system may/may not be in deadlock state.\n\n");

        getch();
        exit(-1);
    }

    printf("\n\nSafe Sequence Found : ");
    for (ll i = 0; i < no_Processes - 1; i++) {
        printf("P%lld-->", safeSeq[i] + 1);
    }
    printf("P%lld", safeSeq[no_Processes - 1] + 1);

}

/*
 Banker's algorithm to find if system is in a safe state or not:
 1) Let Work,Finish,Safe be vectors of length m,n,n respectively.
Work is used to determine current max need
 Initialize: Work = Available
Finish[i] = 0; for i:(1 to n)
2) Find an i such that both
a) Finish[i] = 0
b) Need[i] <= Work
c) push i into Safe
if no such i exists goto step -> 4
3) Work = Work + Allocation[i]
Finish[i] = 1
goto step (2)
4) if Finish [i] = 1 for all i
then the system is in a safe state
5) Print safe.
 */

```

```
bool isSafe() {
    // get safe sequence

    ll work[no_Resources], ind = 0;
    for (ll i = 0; i < no_Resources; i++) work[i] = avail[i];

    ll finish[no_Processes] = {0};
    int count = 0;
    while (count < no_Processes) {
        bool found = false;
        for (ll i = 0; i < no_Processes; i++) {
            if (finish[i] == 0) {
                ll j;
                for (j = 0; j < no_Resources; j++)
                    if (need[i][j] > work[j])
                        break;
                if (j == no_Resources) {
                    for (ll k = 0; k < no_Resources; k++)
                        work[k] += allocated[i][k];

                    safeSeq[count++] = i;

                    finish[i] = 1;

                    found = true;
                }
            }
        }

        if (!found) {
            for (ll i = 0; i < no_Processes; i++) safeSeq[i] = -1;
            return false;
        }
    }

    return true;

}

```

Student Name: AYUSH DWIVEDI
Student ID 11801780
Email Address: akdrishu@gmail.com
GitHub Link: https://github.com/akdrishu/OS_PROJECT
Question-Code: Q3) Write a multithreaded program that implements the banker's algorithm. Create n threads that request and release resources from the bank. The banker will grant the request only if it leaves the system in a safe state. It is important that shared data be safe from concurrent access. To ensure safe access to shared data, you can use mutex locks.

```c
/*Creating multiple processes outside main and applying locks
 * so that shared data be safe from concurrent access*/

void *Code_Processing(void *arg) {
    ll _insafeSeq = *((ll *) arg);

    // lock resources
    pthread_mutex_lock(&lock_Resources);

    // condition check
    while (_insafeSeq != safeSeq[no_completedprocess])
        pthread_cond_wait(&all_condition, &lock_Resources);

    // process
    printf("\n--> Process %lld", _insafeSeq + 1);
    printf("\n\tAllocated : ");
    for (ll i = 0; i < no_Resources; i++)
        printf("%lld ", allocated[_insafeSeq][i]);

    printf("\n\tNeeded    : ");
    for (ll i = 0; i < no_Resources; i++)
        printf("%lld ", need[_insafeSeq][i]);

    printf("\n\tAvailable : ");
    for (ll i = 0; i < no_Resources; i++)
        printf("%lld ", avail[i]);

    printf("\n");
    sleep(1);

    printf("\tResource Allocated!");
    printf("\n");
    sleep(1);
    printf("\tProcess Code Running...");
    printf("\n");
    sleep(5); // process code
    printf("\tProcess Code Completed...");
    printf("\n");
    sleep(1);
    printf("\tProcess Releasing Resource...");
    printf("\n");
    sleep(1);
    printf("\tResource Released!");

    for (ll i = 0; i < no_Resources; i++)
        avail[i] += allocated[_insafeSeq][i];

    printf("\n\tNow Available : ");
    for (ll i = 0; i < no_Resources; i++)
        printf("%lld ", avail[i]);
    printf("\n\n");

    sleep(1);

    // condition broadcast
    no_completedprocess++;
    pthread_cond_broadcast(&all_condition);
    pthread_mutex_unlock(&lock_Resources);
    pthread_exit(NULL);
}
```

Student Name: AYUSH DWIVEDI
Student ID 11801780
Email Address: akdrishu@gmail.com
GitHub Link: https://github.com/akdrishu/OS_PROJECT
Question-Code: Q3) Write a multithreaded program that implements the banker's algorithm. Create n threads that request and release resources from the bank. The banker will grant the request only if it leaves the system in a safe state. It is important that shared data be safe from concurrent access. To ensure safe access to shared data, you can use mutex locks.

```
234
235  /*Funtion for independant process execution one by one. */
236
237  void Process_Execution() {
238
239      printf("\nExecuting Processes...\n\n");
240      sleep(1);
241
242      // run threads
243      pthread_t processes[no_Processes];
244      pthread_attr_t attr;
245      pthread_attr_init(&attr);
246
247      ll processNumber[no_Processes];
248      for (ll i = 0; i < no_Processes; i++) processNumber[i] = i;
249
250      for (ll i = 0; i < no_Processes; i++)
251          pthread_create(&processes[i], &attr, Code_Processing, (void *) (&processNumber[i]));
252
253      for (ll i = 0; i < no_Processes; i++)
254          pthread_join(processes[i], NULL);
255
256      printf("\nAll the Processes have been successfully completed\n");
257
258  }
259
260
261
```

1. DESCRIPTION:

A process in operating systems uses different resources and uses resources in following way.
1) Requests a resource
2) Use the resource
2) Releases the resource

Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

A deadlock occurs if the four hold true. But these conditions are not mutually exclusive.

Mutual Exclusion:

> There should be a resource that can only be held by one process at a time.

Hold and Wait:

> A process can hold multiple resources and still request more resources from other processes which are holding them.

No Preemption:

> A resource cannot be preempted from a process by force. A process can only release a resource voluntarily. In the diagram below, Process 2 cannot preempt Resource 1 from Process 1. It will only be released when Process 1 relinquishes it voluntarily after its execution is complete.

Circular Wait:

Student Name: AYUSH DWIVEDI
Student ID 11801780
Email Address: akdrishu@gmail.com
GitHub Link: https://github.com/akdrishu/OS_PROJECT
Question-Code: Q3) Write a multithreaded program that implements the banker's algorithm. Create n threads that request and release resources from the bank. The banker will grant the request only if it leaves the system in a safe state. It is important that shared data be safe from concurrent access. To ensure safe access to shared data, you can use mutex locks.

A process is waiting for the resource held by the second process, which is waiting for the resource held by the third process and so on, till the last process is waiting for a resource held by the first process. This forms a circular chain.

2. ALGORITHM:

Banker's Algorithm:

1) Let Work,Finish,Safe be vectors of length m,n,n respectively.

Work is used to determine current max need

Initialize: Work = Available

Finish[i] = 0; for i:(1 to n)

2) Find an i such that both

a) Finish[i] = 0

b) Need[i] <= Work

c) push i into Safe

if no such i exists goto step -> 4

3) Work = Work + Allocation[i]

Finish[i] = 1

goto step (2)

4) if Finish [i] = 1 for all i

then the system is in a safe state

5) Print safe.

3. ALGORITHM-CODE:

Overall Complexity is contributed by Banker's algorithm:

CODE	LINE BY LINE COMPLEXITY
bool isSafe() {	O(1)
// get safe sequence	
ll work[no_Resources], ind = 0;	O(1)

Student Name: AYUSH DWIVEDI
Student ID 11801780
Email Address: akdrishu@gmail.com
GitHub Link: https://github.com/akdrishu/OS_PROJECT
Question-Code: Q3) Write a multithreaded program that implements the banker's algorithm. Create n threads that request and release resources from the bank. The banker will grant the request only if it leaves the system in a safe state. It is important that shared data be safe from concurrent access. To ensure safe access to shared data, you can use mutex locks.

for (ll i = 0; i < no_Resources; i++) work[i] = avail[i];	O(no_Resources)
ll finish[no_Processes] = {0},count=0;	O(1)
while (count < no_Processes) {	O(no_Processes)
bool found = false;	O(1)
for (ll i = 0; i < no_Processes; i++) {	O(no_Processes)
if (finish[i] == 0) {	O(1)
ll j;	O(1)
for (j = 0; j < no_Resources; j++)	O(no_Resources)
if (need[i][j] > work[j])	O(1)
break;	O(1)
if (j == no_Resources) {	O(1)
for (ll k = 0; k < no_Resources; k++)	O(no_Resources)
work[k] += allocated[i][k];	O(1)
safeSeq[count++] = i;	O(1)
finish[i] = 1;	O(1)
found = true;	O(1)
}	
}	
}	
if (!found) {	O(1)
for (ll i = 0; i < no_Processes; i++) safeSeq[i] = -1;	O(no_Processes)
return false;	O(1)
}}	
return true;}	O(1)

Student Name: AYUSH DWIVEDI
Student ID 11801780
Email Address: akdrishu@gmail.com
GitHub Link: https://github.com/akdrishu/OS_PROJECT
Question-Code: Q3) Write a multithreaded program that implements the banker's algorithm. Create n threads that request and release resources from the bank. The banker will grant the request only if it leaves the system in a safe state. It is important that shared data be safe from concurrent access. To ensure safe access to shared data, you can use mutex locks.

OVERALL COMPLEXITY → O(no_Processes * no_Processes * no_Resources)

4. CODE SNIPPET FOR CONSTRAINTS:

The only constraint given in question is that the banker will grant the request only if it leaves the system in a safe state. It is important that shared data be safe from concurrent access. To ensure safe access to shared data, you can use mutex locks:

```
173   /*Creating multiple processes outside main and applying locks
174    * so that shared data be safe from concurrent access*/
175
176   void *Code_Processing(void *arg) {
177       ll _insafeSeq = *((ll *) arg);
178
179       // lock resources
180       pthread_mutex_lock(&lock_Resources);
181
182       // condition check
183       while (_insafeSeq != safeSeq[no_completedprocess])
184           pthread_cond_wait(&all_condition, &lock_Resources);
185
186       // process
187       printf("\n--> Process %lld", _insafeSeq + 1);
188       printf("\n\tAllocated : ");
189       for (ll i = 0; i < no_Resources; i++)
190           printf("%lld ", allocated[_insafeSeq][i]);
191
192       printf("\n\tNeeded    : ");
193       for (ll i = 0; i < no_Resources; i++)
194           printf("%lld ", need[_insafeSeq][i]);
195
196       printf("\n\tAvailable : ");
197       for (ll i = 0; i < no_Resources; i++)
198           printf("%lld ", avail[i]);
199
200       printf("\n");
201       sleep(1);
202
203       printf("\tResource Allocated!");
204       printf("\n");
205       sleep(1);
206       printf("\tProcess Code Running...");
207       printf("\n");
208       sleep(5); // process code
209       printf("\tProcess Code Completed...");
210       printf("\n");
211       sleep(1);
212       printf("\tProcess Releasing Resource...");
213       printf("\n");
214       sleep(1);
215       printf("\tResource Released!");
216
217       for (ll i = 0; i < no_Resources; i++)
218           avail[i] += allocated[_insafeSeq][i];
219
220       printf("\n\tNow Available : ");
221       for (ll i = 0; i < no_Resources; i++)
222           printf("%lld ", avail[i]);
223       printf("\n\n");
224
225       sleep(1);
226
227       // condition broadcast
228       no_completedprocess++;
229       pthread_cond_broadcast(&all_condition);
230       pthread_mutex_unlock(&lock_Resources);
231       pthread_exit(NULL);
232   }
233
```

Student Name: AYUSH DWIVEDI
Student ID 11801780
Email Address: akdrishu@gmail.com
GitHub Link: https://github.com/akdrishu/OS_PROJECT
Question-Code: Q3) Write a multithreaded program that implements the banker's algorithm. Create n threads that request and release resources from the bank. The banker will grant the request only if it leaves the system in a safe state. It is important that shared data be safe from concurrent access. To ensure safe access to shared data, you can use mutex locks.

5. ADDITIONAL ALGORITHM:

The only additional algorithm used is for preventing shared data from concurrent access done by using mutex locks. Its code is already mentioned above.

6. BOUNDARY CONDITIONS:

There are five major conditions(boundary) in which this code runs:

a) All the values must be positive, otherwise the code will terminate;

b) Max_required[i][j] > allocated[i][j], for all corresponding i, j; otherwise need[i][j] will become negative and the program will terminate.

c) As per the coding structure, for efficient memory management and avoiding wastage of memory, dynamic memory allocation is used with a maximum value("mxn" in the code) = 100(1e2), hence the total no of processes and total no of resources both should be less than 100, otherwise the program will take too much memory and array out of bound (or segmentation error) RUN-TIME ERRORS will occur.

d) The system is in a safe state, hence here a safe sequence is generated and individual processes are run one by one.

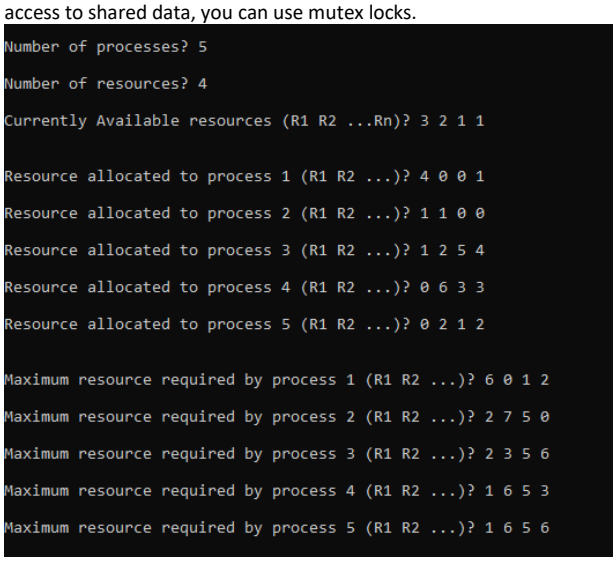
e) The system goes into an unsafe state, hence here the user is informed that the system goes into an unsafe state and hence deadlock wasn't avoided by Banker's algorithm and the program ends.

7. TEST-CASES:

a) Test case 1:

Student Name: AYUSH DWIVEDI
Student ID 11801780
Email Address: akdrishu@gmail.com
GitHub Link: https://github.com/akdrishu/OS_PROJECT
Question-Code: Q3) Write a multithreaded program that implements the banker's algorithm. Create n threads that request and release resources from the bank. The banker will grant the request only if it leaves the system in a safe state. It is important that shared data be safe from concurrent access. To ensure safe access to shared data, you can use mutex locks.

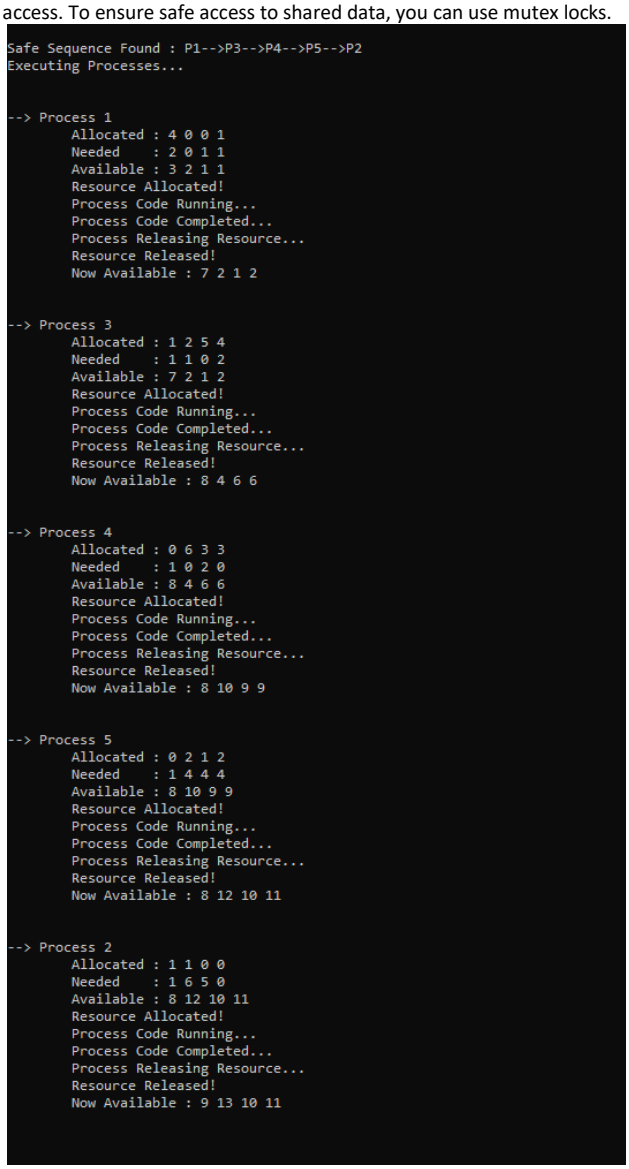
INPUT:



https://github.com/akdrishu/OS_PROJECT/blob/master/input1.PNG

OUTPUT:

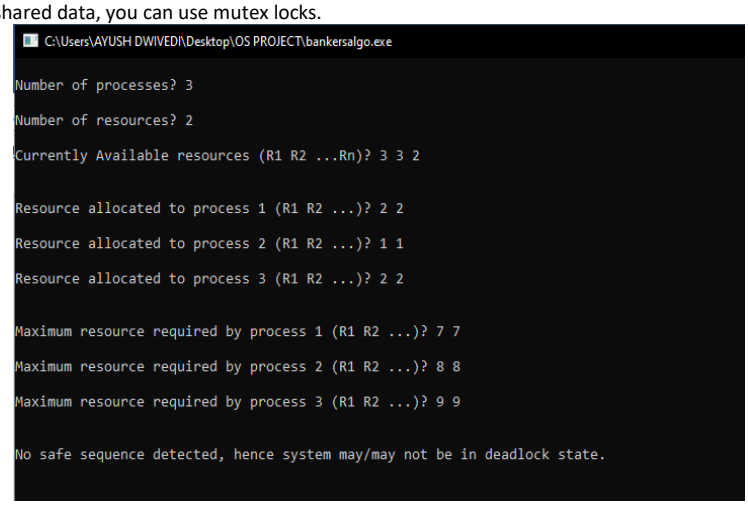
Student Name: AYUSH DWIVEDI
Student ID 11801780
Email Address: akdrishu@gmail.com
GitHub Link: https://github.com/akdrishu/OS_PROJECT
Question-Code: Q3) Write a multithreaded program that implements the banker's algorithm. Create n threads that request and release resources from the bank. The banker will grant the request only if it leaves the system in a safe state. It is important that shared data be safe from concurrent access. To ensure safe access to shared data, you can use mutex locks.



https://github.com/akdrishu/OS_PROJECT/blob/master/testcase1.PNG

b) Test case 2:

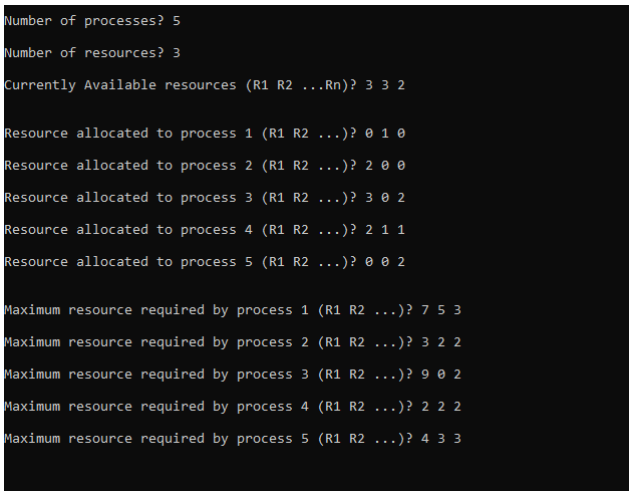
Student Name: AYUSH DWIVEDI
Student ID 11801780
Email Address: akdrishu@gmail.com
GitHub Link: https://github.com/akdrishu/OS_PROJECT
Question-Code: Q3) Write a multithreaded program that implements the banker's algorithm. Create n threads that request and release resources from the bank. The banker will grant the request only if it leaves the system in a safe state. It is important that shared data be safe from concurrent access. To ensure safe access to shared data, you can use mutex locks.



INPUT/OUTPUT:

https://github.com/akdrishu/OS_PROJECT/blob/master/testcase2.PNG

c) Test case 3:



INPUT:

https://github.com/akdrishu/OS_PROJECT/blob/master/input3.PNG

Student Name: AYUSH DWIVEDI
Student ID 11801780
Email Address: akdrishu@gmail.com
GitHub Link: https://github.com/akdrishu/OS_PROJECT
Question-Code: Q3) Write a multithreaded program that implements the banker's algorithm. Create n threads that request and release resources from the bank. The banker will grant the request only if it leaves the system in a safe state. It is important that shared data be safe from concurrent access. To ensure safe access to shared data, you can use mutex locks.

```
Safe Sequence Found : P2-->P4-->P5-->P1-->P3
Executing Processes...


--> Process 2
        Allocated : 2 0 0
        Needed    : 1 2 2
        Available : 3 3 2
        Resource Allocated!
        Process Code Running...
        Process Code Completed...
        Process Releasing Resource...
        Resource Released!
        Now Available : 5 3 2


--> Process 4
        Allocated : 2 1 1
        Needed    : 0 1 1
        Available : 5 3 2
        Resource Allocated!
        Process Code Running...
        Process Code Completed...
        Process Releasing Resource...
        Resource Released!
        Now Available : 7 4 3


--> Process 5
        Allocated : 0 0 2
        Needed    : 4 3 1
        Available : 7 4 3
        Resource Allocated!
        Process Code Running...
        Process Code Completed...
        Process Releasing Resource...
        Resource Released!
        Now Available : 7 4 5


--> Process 1
        Allocated : 0 1 0
        Needed    : 7 4 3
        Available : 7 4 5
        Resource Allocated!
        Process Code Running...
        Process Code Completed...
        Process Releasing Resource...
        Resource Released!
        Now Available : 7 5 5


--> Process 3
        Allocated : 3 0 2
        Needed    : 6 0 0
        Available : 7 5 5
        Resource Allocated!
        Process Code Running...
        Process Code Completed...
        Process Releasing Resource...
        Resource Released!
        Now Available : 10 5 7
```

OUTPUT:

https://github.com/akdrishu/OS_PROJECT/blob/master/testcase3.PNG

Student Name: AYUSH DWIVEDI
Student ID 11801780
Email Address: akdrishu@gmail.com
GitHub Link: https://github.com/akdrishu/OS_PROJECT
Question-Code: Q3) Write a multithreaded program that implements the banker's algorithm. Create n threads that request and release resources from the bank. The banker will grant the request only if it leaves the system in a safe state. It is important that shared data be safe from concurrent access. To ensure safe access to shared data, you can use mutex locks.

8. GitHub Revisions:

A total of **21 different GitHub COMMITS** were made during the project, but here I will list the links of the 5 prominent ones:

1.) https://github.com/akdrishu/OS_PROJECT/commit/4a7a3d2611bc0162757e612b8ca90f1b28b3daf8 : shows simple implementation of banker's algorithm .

2.) https://github.com/akdrishu/OS_PROJECT/commit/232bd734d6e4c93abe3e10b0ccf4efc5fe900e27 : shows bankers algorithm with process creation in C-C++ mixed style.

3.) https://github.com/akdrishu/OS_PROJECT/commit/ed81e7084ceaa5d40ff972880d11389eae8542ee : shows the above in only C++ approach using newer functions like mutex.lock().

4.) https://github.com/akdrishu/OS_PROJECT/commit/37a690a36a0e3dbb9b46f1493d2bebbcbe3a7ed9 : shows the above in C++ with polymorphism.

5.) https://github.com/akdrishu/OS_PROJECT/commit/13473209177fa222c0d4906958df91658bc8295d#diff-ace6c4ee712a1406778481e2bd1bf601L117 : the final updated project with all the details completed.