Student Name: Pawan Dubey

Student ID: 11606675

Email ID: [dpawan494@gmail.com](mailto:dpawan494@gmail.com)

GitHub Link: <https://github.com/anny17>

Code:

|  |
| --- |
| #include<stdio.h> |
|  |

|  |
| --- |
| #include<pthread.h> |
|  |

|  |
| --- |
| #include<stdlib.h> |
|  |

|  |
| --- |
| #include<unistd.h> |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| pthread\_mutex\_t lock; |
|  |

|  |
| --- |
| int Numofprocess; |
|  |

|  |
| --- |
| int Numofresources; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| int Allocation[10][10]; |
|  |

|  |
| --- |
| int Max[10][10]; |
|  |

|  |
| --- |
| int Need[10][10]; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| int Available[10]; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| int checksafe(int required[],int pro) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| //////Declaration of duplicates |
|  |

|  |
| --- |
| int Allocation1[10][10]; |
|  |

|  |
| --- |
| int Need1[10][10]; |
|  |

|  |
| --- |
| int Available1[10]; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /////Initialiging the duplicates |
|  |

|  |
| --- |
| //Allocation |
|  |

|  |
| --- |
| for(int i=0;i<Numofprocess;i++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| for(int j=0;j<Numofresources;j++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| Allocation1[i][j] = Allocation[i][j]; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| //Available |
|  |

|  |
| --- |
| for(int j=0;j<Numofresources;j++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| Available1[j] = Available[j]; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /////////After Asigning the reqired resources state of the machine |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| if(pro != -1){ |
|  |

|  |
| --- |
| for(int j=0;j<Numofresources;j++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| Allocation1[pro][j] = Allocation1[pro][j] + required[j]; |
|  |

|  |
| --- |
| Available1[j] = Available1[j] - required[j]; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| //Need |
|  |

|  |
| --- |
| for(int i=0;i<Numofprocess;i++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| for(int j=0;j<Numofresources;j++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| Need1[i][j] = Max[i][j] - Allocation1[i][j]; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /////printing all the values |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| printf("Available resources:\n"); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| for(int j=0;j<Numofresources;j++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| printf("Resources %d\t%d\n",j,Available1[j]); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| printf("Max matrix:\n"); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| for(int i=0;i<Numofprocess;i++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| for(int j=0;j<Numofresources;j++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| printf("%d\t",Max[i][j]); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| printf("\n"); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| printf("Alocation matrix:\n"); |
|  |

|  |
| --- |
| for(int i=0;i<Numofprocess;i++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| for(int j=0;j<Numofresources;j++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| printf("%d\t",Allocation1[i][j]); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| printf("\n"); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| printf("Need matrix:\n"); |
|  |

|  |
| --- |
| for(int i=0;i<Numofprocess;i++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| for(int j=0;j<Numofresources;j++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| printf("%d\t",Need1[i][j]); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| printf("\n"); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| //Checking the safe state |
|  |

|  |
| --- |
| int flagforunsafe = 0; |
|  |

|  |
| --- |
| int arrayflag[Numofprocess];/\* Array flag declaration and initialization\*/ |
|  |

|  |
| --- |
| for(int i=0;i<Numofprocess;i++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| arrayflag[i] = 0;//If process is not finished yet... it will kept as Zero |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| printf("Everything copied and array flag has been initialized\n"); |
|  |

|  |
| --- |
| /\* K loop because it will take exactly checks equal to number of processes to be found in safe state \*/ |
|  |

|  |
| --- |
| for(int k=0;k<Numofprocess;k++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| printf("running for k=%d\n",k); |
|  |

|  |
| --- |
| //for each time we will check each process to fill it's requirement \*/ |
|  |

|  |
| --- |
| for(int i=0;i<Numofprocess;i++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| printf("running for k=%d and process =%d\n",k,i); |
|  |

|  |
| --- |
| if(arrayflag[i] == 0)//Only if it's not compleated yet |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| printf("process %d was not compleated till %dth value of k\n",i,k); |
|  |

|  |
| --- |
| int flagforneed = 0;//Used to check if need of process i is less than availability of resource instances |
|  |

|  |
| --- |
| for(int j=0;j<Numofresources;j++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| if(Available1[j] < Need1[i][j]) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| printf("Available is less than the need in repetition %d for process %d at resource %d\n",k,i,j); |
|  |

|  |
| --- |
| flagforneed = 1; |
|  |

|  |
| --- |
| break;//If any of resource instanse is less then this process can not be satisfied yet |
|  |

|  |
| --- |
| }else{printf("Available is more or equal than the need in repetition %d for process %d at resource %d\n",k,i,j); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| if(flagforneed == 0)//If process can be satisfied |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| printf("Process %d can be provided all the resources that it needs in the repetition %d\n",i,k); |
|  |

|  |
| --- |
| arrayflag[i] = 1;//process compleated |
|  |

|  |
| --- |
| for(int j=0;j<Numofresources;j++)//Updating the Available Array |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| printf("Updated available instances of resource %d from the repetition %d is %d from %d\n",j,k,Available1[j] + Allocation1[i][j],Available1[j]); |
|  |

|  |
| --- |
| Available1[j] = Available1[j] + Allocation1[i][j]; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| printf("Job done for repetition %d\n it has compleated process %d\n",k,i); |
|  |

|  |
| --- |
| goto next\_k;//go for next k value |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| if(i+1 == Numofprocess) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| flagforunsafe = 1; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| //Come here for next k value |
|  |

|  |
| --- |
| next\_k: printf("going for next k value till the time it was %d\n",k); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return flagforunsafe; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\*A function to be shared by all the n threads created by user in main function |
|  |

|  |
| --- |
| here each thread will work like a process every time a new thread will start excecution it will ask for the |
|  |

|  |
| --- |
| extra resources needed by the thread, if it leads the system in safe state then the changes will be made permanent to the allocation matrix and availability list of instances of each resources\*/ |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void function(int pro) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| pthread\_mutex\_lock(&lock); |
|  |

|  |
| --- |
| int Allocation1[Numofresources]; |
|  |

|  |
| --- |
| printf("Enter request array for process %d\n\n",pro); |
|  |

|  |
| --- |
| for(int j=0;j<Numofresources;j++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| printf("Enter the number of instaces required of resource %d\n",j); |
|  |

|  |
| --- |
| scanf("%d",&Allocation1[j]); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| if (checksafe(Allocation1,pro) == 1) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| printf("Request can not be granted\n"); |
|  |

|  |
| --- |
| }else |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| printf("Request granted :\n"); |
|  |

|  |
| --- |
| //Update the allocation, Need, Maximum required matrix |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| pthread\_mutex\_unlock(&lock); |
|  |

|  |
| --- |
| pthread\_exit(NULL); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| //////Print function |
|  |

|  |
| --- |
| void print() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| printf("Available resources:\n"); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| for(int j=0;j<Numofresources;j++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| printf("Resources %d\t%d\n",j,Available[j]); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| printf("Max matrix:\n"); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| for(int i=0;i<Numofprocess;i++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| for(int j=0;j<Numofresources;j++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| printf("%d\t",Max[i][j]); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| printf("\n"); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| printf("Alocation matrix:\n"); |
|  |

|  |
| --- |
| for(int i=0;i<Numofprocess;i++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| for(int j=0;j<Numofresources;j++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| printf("%d\t",Allocation[i][j]); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| printf("\n"); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| printf("Need matrix:\n"); |
|  |

|  |
| --- |
| for(int i=0;i<Numofprocess;i++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| for(int j=0;j<Numofresources;j++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| printf("%d\t",Max[i][j] - Allocation[i][j]); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| printf("\n"); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| int main() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| pthread\_mutex\_init(&lock,NULL); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| printf("Enter number of processes:\n"); |
|  |

|  |
| --- |
| scanf("%d",&Numofprocess); |
|  |

|  |
| --- |
| fflush(stdin); |
|  |

|  |
| --- |
| printf("Enter number of Resources:\n"); |
|  |

|  |
| --- |
| scanf("%d",&Numofresources); |
|  |

|  |
| --- |
| fflush(stdin); |
|  |

|  |
| --- |
| printf("Enter the allocation matrix:\n"); |
|  |

|  |
| --- |
| for(int i=0;i<Numofprocess;i++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| for(int j=0;j<Numofresources;j++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| printf("Enter the number of instances of resource %d process %d have:\n",j,i); |
|  |

|  |
| --- |
| scanf("%d",&Allocation[i][j]); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| printf("Enter the Maximum need matrix:\n"); |
|  |

|  |
| --- |
| for(int i=0;i<Numofprocess;i++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| for(int j=0;j<Numofresources;j++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| printf("Enter the number of instances of resource %d process %d Need at max:\n",j,i); |
|  |

|  |
| --- |
| scanf("%d",&Max[i][j]); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| int Allocation1[Numofresources]; |
|  |

|  |
| --- |
| for(int j=0;j<Numofresources;j++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| printf("Enter the number of instances of resource %d have\n",j); |
|  |

|  |
| --- |
| scanf("%d",&Available[j]); |
|  |

|  |
| --- |
| Allocation1[j] = 0; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| int f = checksafe(Allocation1,-1); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| if(f==1){printf("already not in safe state\n"); |
|  |

|  |
| --- |
| return 0;} |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| pthread\_t proc[Numofprocess]; |
|  |

|  |
| --- |
| for(int i=0;i<Numofprocess;i++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| pthread\_create(&proc[i],NULL,function,i); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| printf("All threads created\n"); |
|  |

|  |
| --- |
| for(int i=0;i<Numofprocess;i++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| pthread\_join(proc[i],NULL); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| return 0; |
|  |

|  |
| --- |
| } |
|  |

Question 1: Explain the problem in terms of operating system concept? (Max 200 word)

Answer:

First concept used in this program is threading.

I have created a multithreaded program in which each thread is acting as a process and requesting for some resources, and using banker’s algorithm to check if allowing that much resources leaves the system in safe state or not.

Used mutex lock to keep each thread isolated while calculating the result for their requested resources.

Question 2. Write the algorithm for proposed solution of the assigned problem.

Answer:

First take the number of processes and number of resources

Now take the present state of our system that is Allocation matrix

And take the Maximum need matrix. (Which is maximum number of resources a process can request for at any time).

Then I’ve calculated need matrix

And called the banker’s algorithm function to check if system is in safe state or not

If it is not in safe state terminate the program

Now create all the threads and join them

Each thread takes the number of resources to be requested.

Call banker’s algorithm for each test

For each thread print respected results.

Question 3: Calculate complexity of implemented algorithm. (Student must specify complexity of each line of code along with overall complexity)

Answer:

For checksafe function:

#1.

copying each data in local variables with complexity O(r\*p)

r – number of resources

p- number of processes

#2.

updating the Allocation and available matrix according to the request

with complexity O(r)

#3.

Printing all the changed data (Assumed that request is granted) with complexity O(r\*p)

#4.

Creating a flag array with complexity O(p)\

#5.

Checking the safe state with complexity O(p\*p\*r)

For threading function:

#1.

Taking input for needed resources with complexity O(r)

For main function:  
#1.

Taking allocation and need matrix input with complexity O(r\*p)

And available resources with complexity O(r)

#2.

Creating all the threads and joining them with complexity O(p)

Final complexity:  
each thread is calling the checksafe function once the total complexity of checksafe function is O(p\*p\*r)

Hence total complexity of this program if O(p\*p\*p\*r)

Which can be said as complexity O(n\*\*4)

Question 4. Explain all the constraints given in the problem. Attach the code snippet of the implemented constraint.

#1. MULTITHREADED PROGRAM:-

Created and joined multiple thread in my program

|  |
| --- |
| pthread\_t proc[Numofprocess]; |
|  |

|  |
| --- |
| for(int i=0;i<Numofprocess;i++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| pthread\_create(&proc[i],NULL,function,i); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| printf("All threads created\n"); |
|  |

|  |
| --- |
| for(int i=0;i<Numofprocess;i++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| pthread\_join(proc[i],NULL); |
|  |

}

#2. BANKER’S ALGORITHEM:-

Implemented banker’s algorithm

|  |
| --- |
| if (checksafe(Allocation1,pro) == 1) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| printf("Request can not be granted\n"); |
|  |

|  |
| --- |
| }else |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| printf("Request granted :\n"); |
|  |

|  |
| --- |
| //Update the allocation, Need, Maximum required matrix |
|  |

}

#3. MUTEX LOCK:-

Used mutex lock to keep each thread isolated while checking for safestate

|  |
| --- |
| void function(int pro) |
|  |

|  |
| --- |
| { |
|  |

pthread\_mutex\_lock(&lock);

// code

|  |
| --- |
| pthread\_mutex\_unlock(&lock); |
|  |

|  |
| --- |
| pthread\_exit(NULL); |
|  |

}

Question 5. If you have implemented any additional algorithm to support the solution, explain

the need and usage of the same

No I’ve not implemented any extra algorithm

Question 6. Explain the boundary conditions of the implemented code.

#1.

This code will check for safe state at very first time when you will enter the allocation, maximum matrix and available resources.

#2.

It will check for safe state for each resource request, if updated system is in safe state it will say you can allow the request (But it won’t allow)

#3.

If it leads to an unsafe state it will say you can’t allow the request

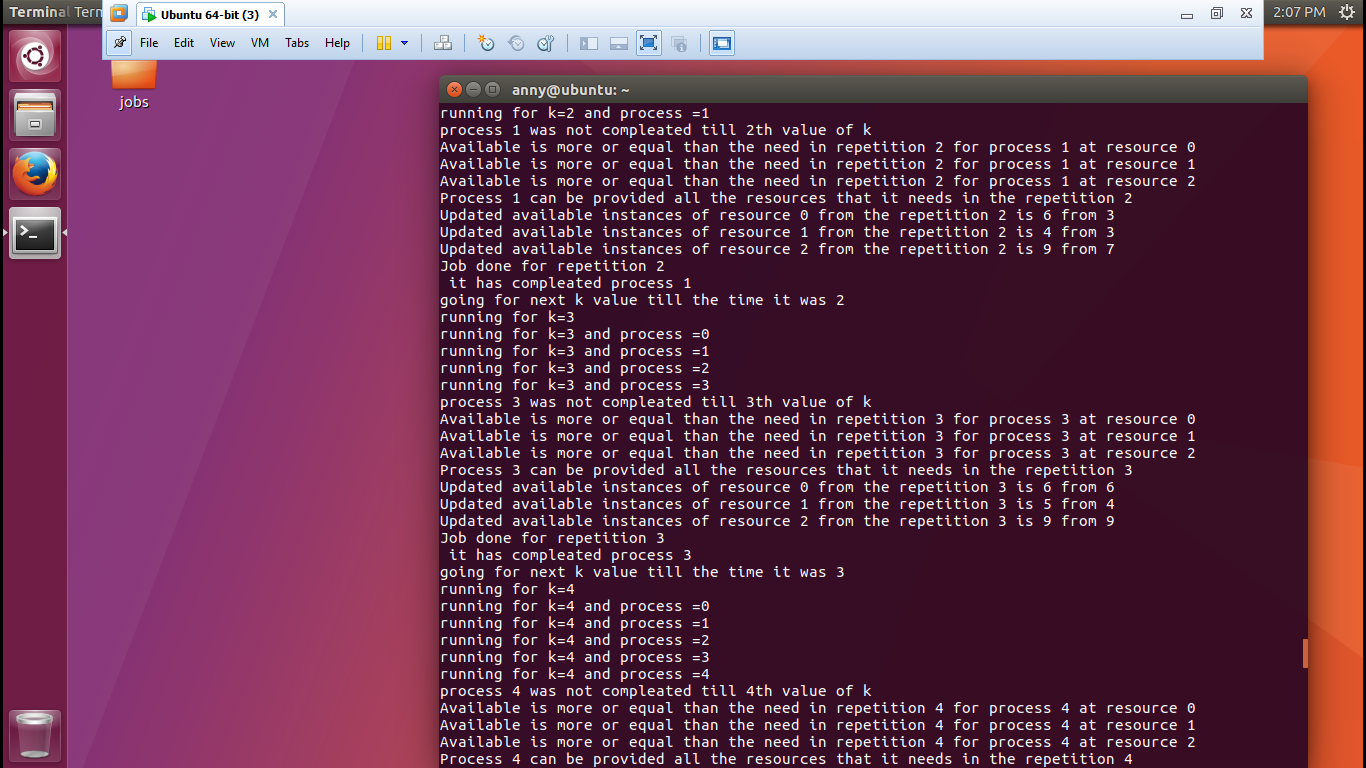
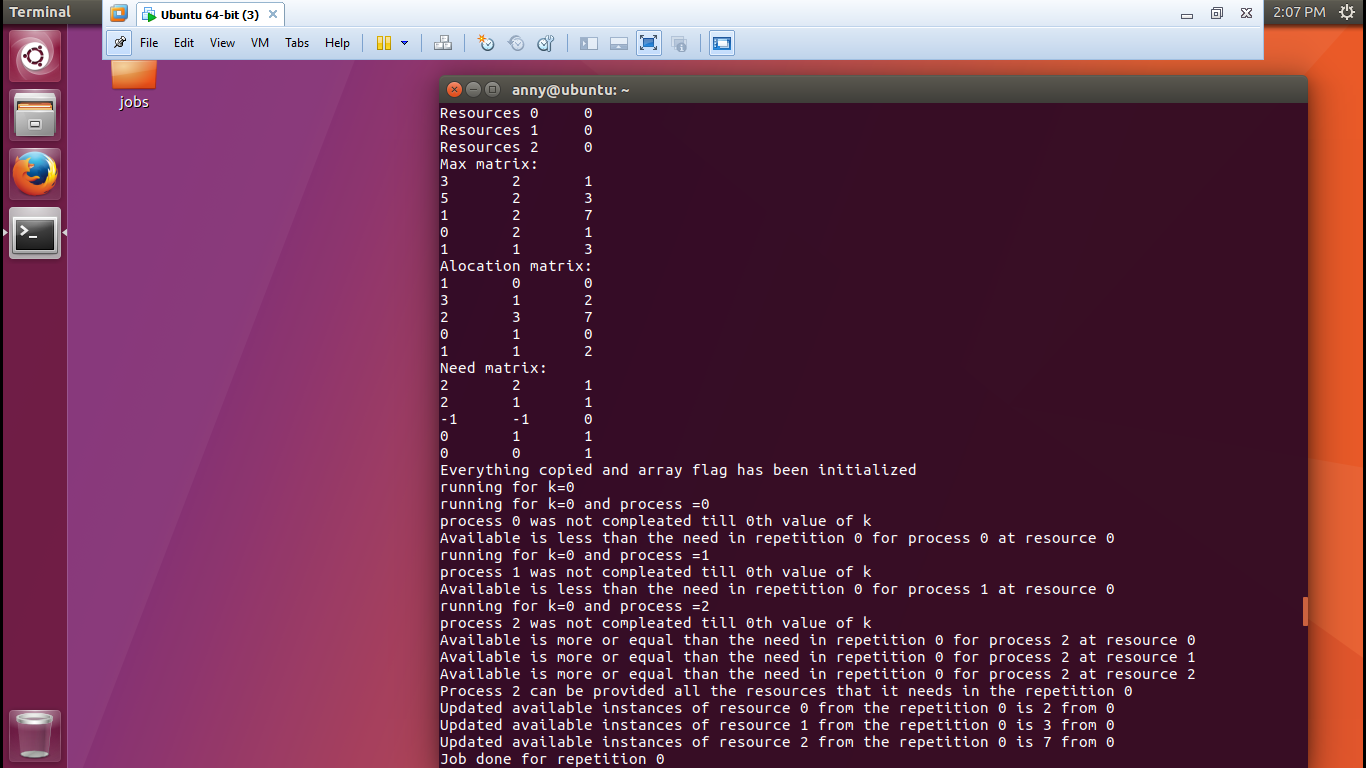
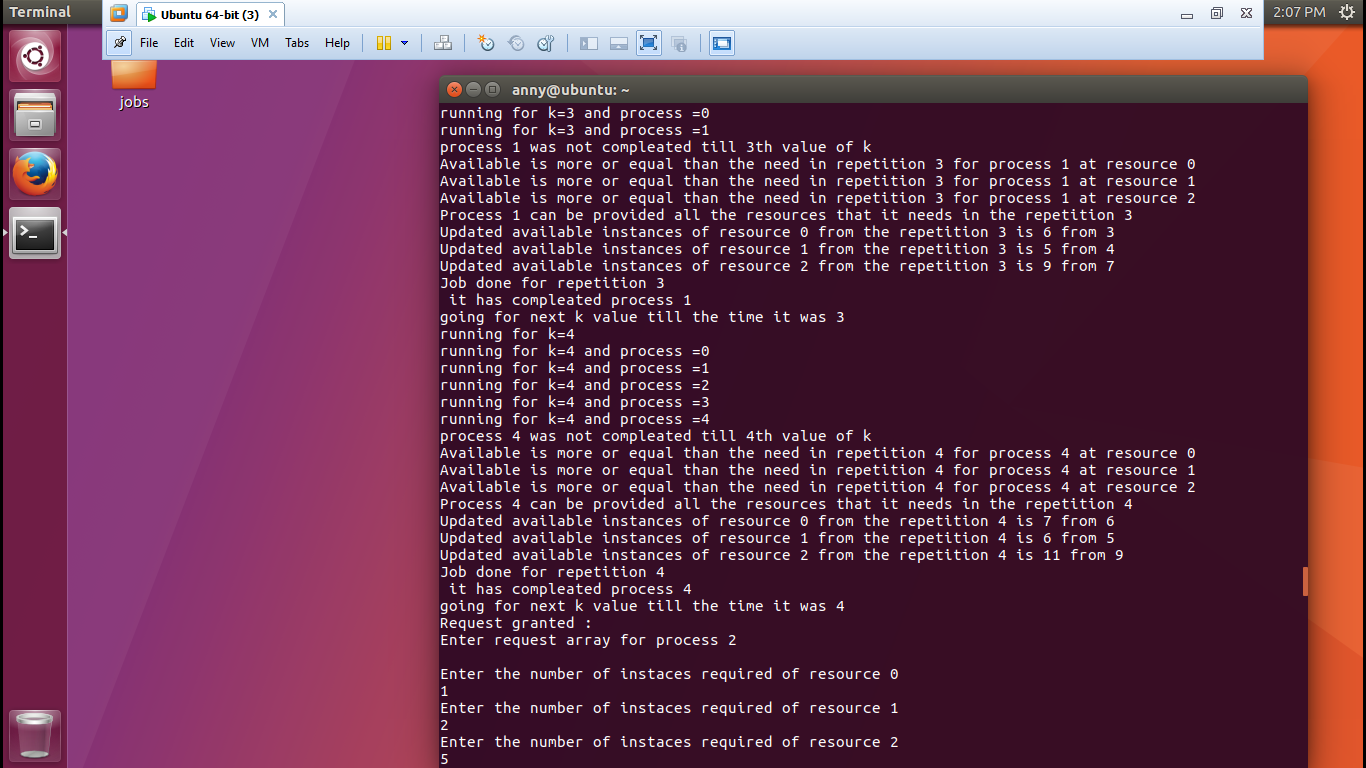
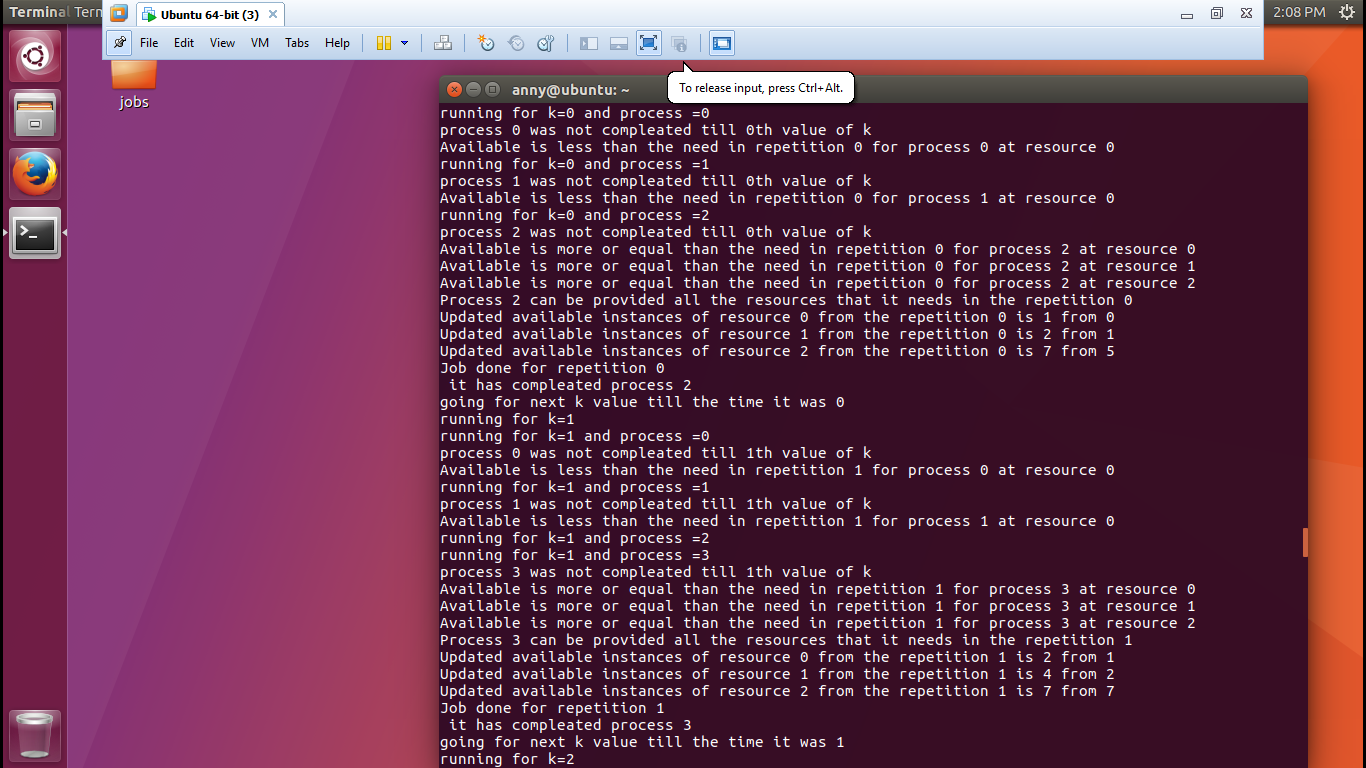
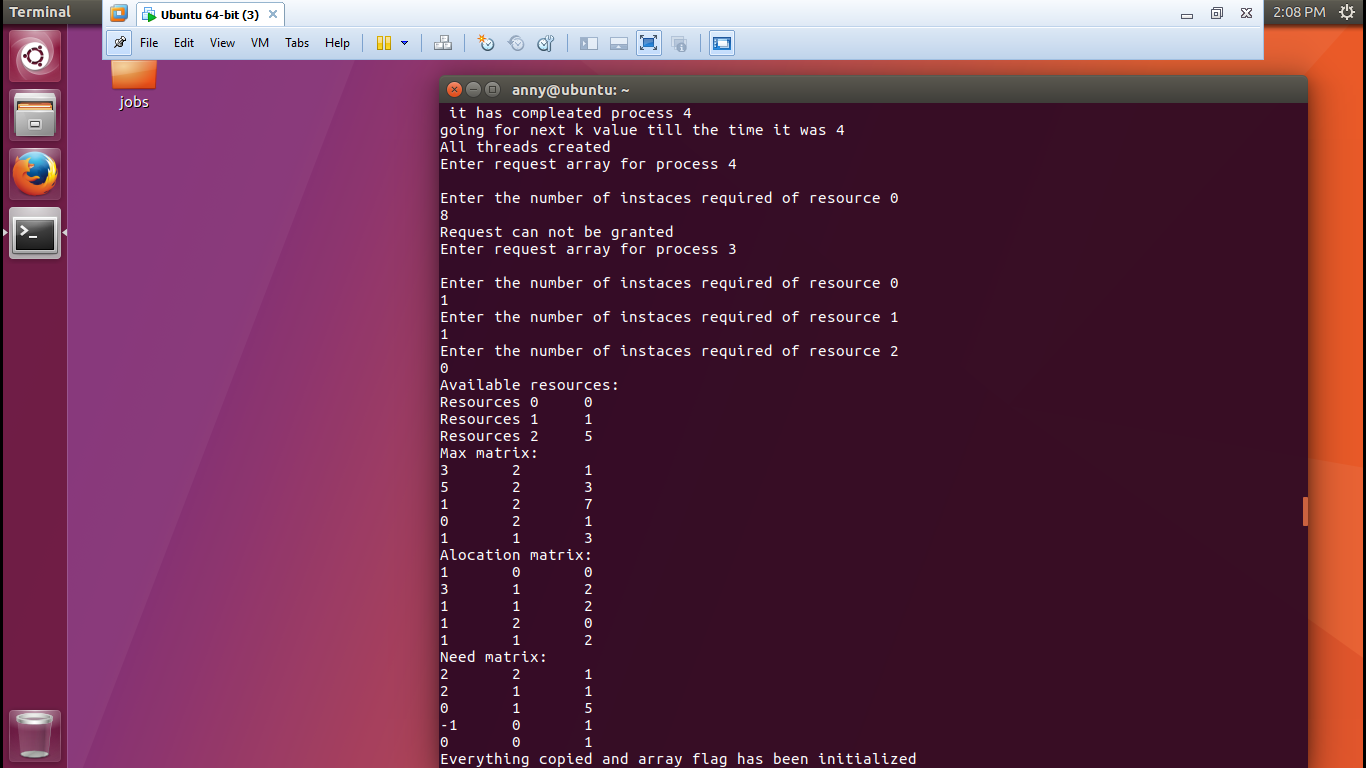
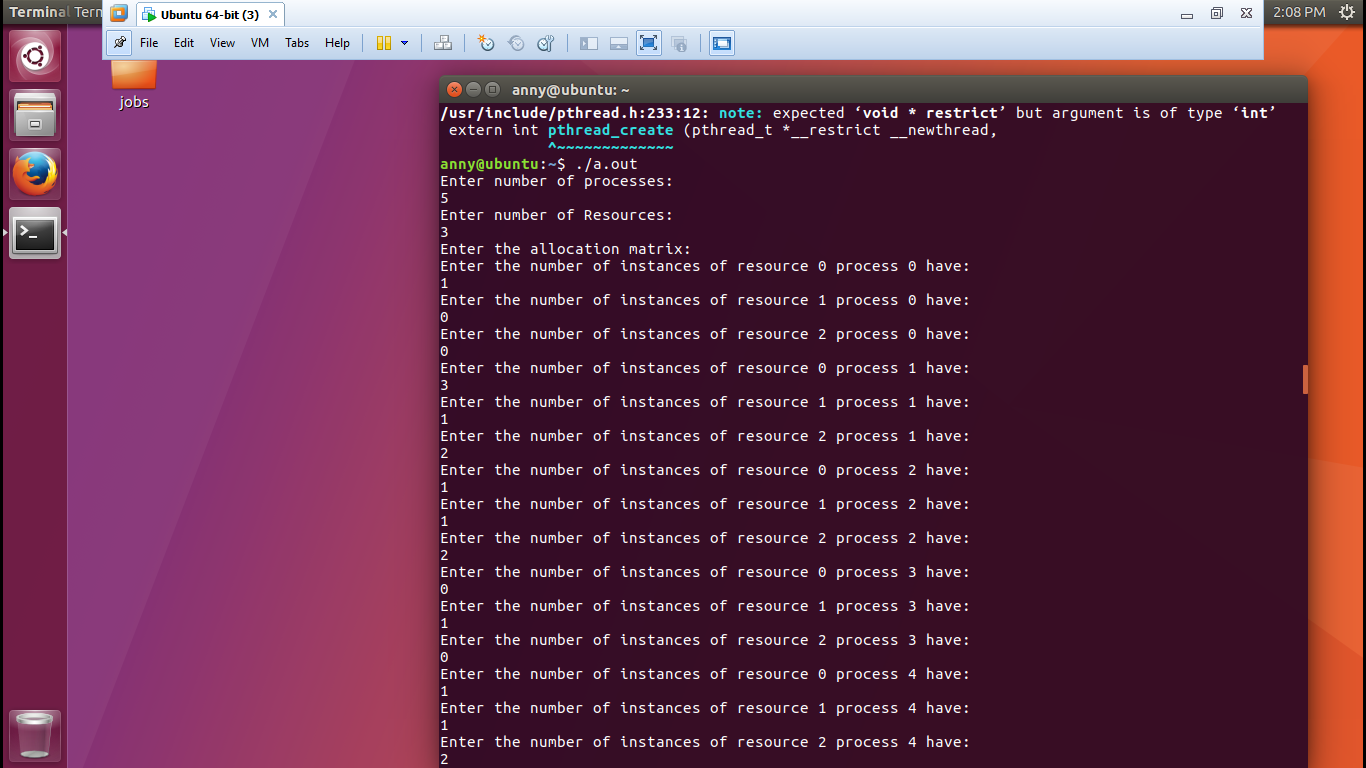
It basically happens when the request gets larger or resources get lesser

Question 7. Explain all the test cases applied on the solution of assigned problem.

All the explanation and flow of program is written in the code itself

It gives a brief idea how things are going inside the program

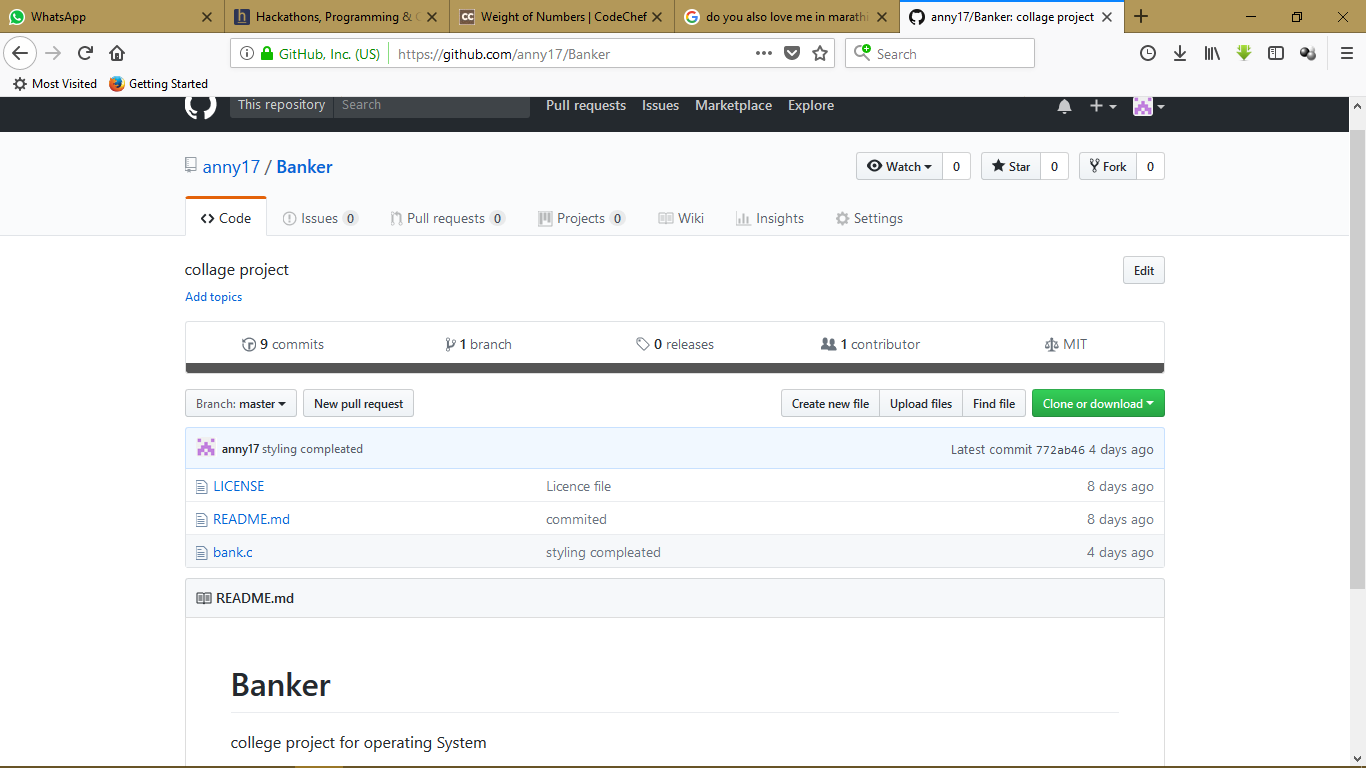
I have attached the test run of my program



Question 8. Have you made minimum 5 revisions of solution on GitHub?

GitHub Link:

https://github.com/anny17/Banker



Thank you Sir!