Student Name: Rick Dhillon

Student Id: 11801616

Email Address: [rickdhillon2000@gmail.com](mailto:rickdhillon2000@gmail.com)

GitHub Link: <https://github.com/rickdhillon31/Rick-Dhillon/tree/master>

Code: 3

DESCRIPTION

The question is to write a multi-threaded program that implements banker’s algorithms and check whether the system is in safe state or not. And grant the resources if it is in safe state.

Multi-threaded means that the program can handle more than one user at a single instance of time, or in other words the system can fulfill requests concurrently. It can manage multiple requests from the user without having to multiple copies of the program running.

Banker’s algorithm is a resource an allocation and deadlock avoidance algorithm. It is used for resources allocation to different users or threads. The main function of this banker’s algorithm is to check if the system is in safe state or not if it is in safe state allocate the resources according to the request and then take back the resources back when done.

ALGORITHM

First threads are created. Now, let there be n threads and m resources then thse arrays are initialized as follow: -

* Available: 1-D array with size m, indicating the total number of resources system have.
* Max: it is 2-D array with size n\*m that defines the max demand of each process of each resource.
* Allocation: it is a 2-D array of size n\*m the that defines the total number of resources of each type currently allocated to each process.
* Need: It is a 2-D array of size n\*m that tells the remaining resource need of each process.

This algorithm is divided into two parts, first is checking the safety and second is resource request.

Safety algorithm-

1. Initializes all the task with flag such that work is not completed.
2. Find the task for which work is not finished and the need for that request is less than what we have available.
   1. If no such step go to step(4)
3. Available=Available + allocation
   1. Mark that task as finished
   2. Go to step 2
4. If there is no such task which is incomplete than the system is in safe state.

Resource-request Algorithm-

1. If request <=need for any process go to step 2 else raise error
2. If request <=available go to step 3 else the process must wait since resources are not available at this current moment.
3. Allocate the resources to the process and modify the following variables
   1. Available = available – request
   2. Allocation = allocation + request
   3. Need = need – request

COMPLEXITY

n= no of processes

m= no of resources

For safety algorithm the complexity is=> o(n\*m)

For resource request algorithm the complexity is=>o(n)

Overall complexity is=> o(n\*n\*m)

CONSTRAINTS

* NUMBER OF CUSTOMERS- total number of threads
  + #define NUMBEROFCUSTOMERS 5
* NUMBER OF RESOURCES- no of resources
  + #define NUMBEROFRESOURCES 3
* available- total number of instance resources which are available
  + int available [NUMBEROFRESOURCES];
* maximum- this is the total maximum no of resources that a process can request
  + int maximum[NUMBEROFCUSTOMERS][NUMBEROFRESOURCES];
* allocation- this is the total no of resources that are already allocate to a particular process.
  + int allocation[NUMBEROFCUSTOMERS][NUMBEROFRESOURCES];
* need- this is the number of resources that are still require even after allocation
  + int need[NUMBEROFCUSTOMERS][NUMBEROFRESOURCES];
* tid- it is the thread
  + pthread\_t tid[NUMBEROFCUSTOMERS]; /\* the thread identifiers \*/

Boundary condition

The boundary conditions for this code is that the system should be in safe state all the time and there should not be any deadlock condition. As soon as deadlock appears, or the system runs into the unsafe state or in other word there is no safe sequence then the program will abort. These are the 2 conditions that needs to be fulfilled all the time for this program to run.

Test cases

* Test case 1-
  + ./main 5 6
  + Not enough resources
* Test case 2-
  + ./main 5 -5 5
  + Argument -5 must be non-negative
* Test case 3-
  + ./main 5 -3
  + Not enough resources
* Test case 4-
  + ./main 5 5 5
  + Program will get stuck
* Test case 4-
  + ./main 20 20 20
  + Program will run successfully.