**Name** : Siddhesh Srikant Dhainje

**Roll no.** : 33221

**Assignment** : 5

**Statement** :

Implement the C program for Deadlock Avoidance Algorithm: Banker’s Algorithm

**Theory** :

* What is a Deadlock?
* In a multiprogramming environment, several processes may compete for finite number of resources.
* A process requests resources; if the resources are not available at that time, the process enters a waiting state.
* Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes.
* This situation is called a deadlock.
* System Model
* A system consists of a finite number of resources to be distributed among a number of competing processes.
* The resources are partitioned into several types, each consisting of some number of identical instances.
* Memory space, CPU cycles, files, and I/O devices are examples of resource types.
* If a system has to CPUs, then the resource type CPU has two instances.
* If a process requests an instance of a resource type, the allocation of any instance type will satisfy the request.
* If it will not, then the instances are not identical, and the resource type classes have not been defined properly
* A process must request a resource before using it and must release the resource after using it.
* A process may request as many resources as it requires to carry out its designated task
* Under the normal mode of operation, a process may utilize a resource in the following sequence:
* Request:
* The process requests the resource.
* If the request cannot be granted immediately (for example, if the resource is being used by another process), then the requesting process must wait until it can acquire the resources.
* Use:
* The process can operate on the resource (for example, if the resource is a printer, the process can print on the printer)
* Release:
* The process releases the resource
* The request and release of resources are system calls.
* Examples are the request() and release() device, open() and close() file, and allocate() and free() memory system calls.
* Request and release of resources that are not managed by the operating system can be accomplished through the wait() and signal() operations on semaphores or through wait() and signal() operations on the semaphores or through acquisition and release of a mutex lock.
* Deadlock Avoidance
* Deadlock avoidance requires that the operating system be given in advance additional information concerning which resources a process will request and use during its lifetime
* With this additional knowledge, it can decide for each request whether or not the process should wait.
* To decide whether the current request an be satisfied or must be delayed, the system must consider the resources currently available, the resources currently allocated to each process, and the future requests and releases of each process.
* If a system does not employ either a deadlock-prevention or a deadlock-avoidance algorithm, then a deadlock situation may arise.
* In this environment, the system can provide an algorithm that examines the state of the system to determine whether a deadlock has occurred and an algorithm to recover from the deadlock.
* Banker’s Algorithm
* The banker’s algorithm, as the name suggests, could be used in the banking system to ensure that the bank never allocated its available cash in such a way that it could no longer satisfy the needs of all its customers.
* The following data structures are needed for the banker’s algorithm where n is the number of processes and m is the number of resources.
* Available:
* A vector of length m indicates the number of available resources of each type.
* If Available[j] equals k, then k instances of resources type Rj are available.
* Max:
* An n x m matrix defines the maximum demand of each process.
* If Max[i][j] equals k, then process Pi may request at most k instances of the resource type Rj.
* Allocation:
* An n x m matrix defines the maximum demand of each process.
* If Allocation[i][j] equals k, then process Pi is currently allocated k instances of resource type Rj.
* Need:
* An n x m matrix indicates the remaining resource need of each process.
* If Need[i][j] equals k, then process Pi may need k more instances of resource type Rj to complete its task.
* Note that Need[i][j] = Max[i][j] – Allocation[i][j]
* Safety Algorithm
* The safety algorithm is used for finding out whether or not a system is in a safe state.
* Let Work and Finish be vectors of length m and n, respectively. Initialize Work = Available and Finish[i] = false

for i = 0, 1, ..., n - 1.

* Find an index i such that both
* Finish[i] == false
* Needi <= Work

If no such i exists, go to step 4.

* Work = Work + Allocation;
* Finish[i] = true
* Go to step 2.
* If Finish[i] ==true for all i, then the system is in a safe state.
* Resource - Request Allocation
* Let Requesti be the request vector for process Pi.
* If Requesti [j] == k, then process Pi wants k instances of resource type Rj.
* When a request for resources is made by process Pi, the following actions are taken:
* If Requesti <= Needi, go to step 2. Otherwise, raise an error condition, since the process has exceeded its maximum claim.
* If Requesti <= Available, go to step 3. Otherwise, Pi must wait, since the resources are not available.
* Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows:

Available = Available – Requesti

Allocationi =Allocationi +Requesti

Needi = Needi – Requesti

* If the resulting resource-allocation state is safe, the transaction is completed, and process Pi is allocated its resources.
* However, if the new state is unsafe, then Pi must wait for Requesti , and the old resource-allocation state is restored.

Screenshots of Output :









Conclusion :

* In this assignment, I have successfully executed the Banker’s Algorithm which is used for Deadlock Avoidance.