Group A
Experiment No. 4

Title :

Aim: **To implement** Banker's algorithm for deadlock avoidance and detection.

THEORY:

DEADLOCK: A deadlock involving a set of processes D is a situation in which

1. Every process p_i in D is blocked on some event e_i.
2. Event e_i can only be caused by some process (es) in D.

A long delay in the occurrence of an awaited event , i.e. starvation of some kind ,cannot contribute to a deadlock. . The important requirement is that processes capable of causing an event must themselves belong to D. This makes it impossible for the event to happen.

Deadlock affects the progress of processes by causing indefinite delays in resource allocation. Such delays have serious consequences for the response times of processes, idling and wastage of resources allocated to processes, and the performance of the system.

HANDLING DEADLOCKS:

Two fundamental approaches used for handling deadlocks are:

1. Detection and resolution of deadlocks.
2. Avoidance of deadlocks.

In the former approach, the OS detects deadlock situations as and when they arise. It then performs some actions aimed at ensuring progress for some of the deadlocked processes. These actions constitute deadlock resolution. The latter approach focuses on on avoiding the occurrence of deadlocks. This approach involves checking each resource request to ensure that granting does not lead to deadlock. The detection and resolution approach does not perform any such checks.

Banker's algorithm

The **Banker's algorithm** is a resource allocation & deadlock avoidance algorithm developed by Edsger Dijkstra that tests for safety by simulating the allocation of pre-determined maximum possible amounts of all resources, and then makes a "safe-state" check to test for possible deadlock conditions for all other pending activities, before deciding whether allocation should be allowed to continue.

The Banker's algorithm uses an avoidance policy. Hence no validity constraint is specified, and processes are free to make any resource requests. The system uses two tests – *feasibility test* and *safety test.* – to govern its resource allocation actions when process p_j makes a request $Req_{j,k}$ for units of resource class k.

Feasible request : A feasible request if, after granting the request, the allocated resources of class k do not exceed the total number of resource units of class k in the system.

Safe request : A feasible request $Req_{j,k}$ is a safe request at time t if, after granting The request, there exists at least one sequence of allocations and releases by which all processes in the system can complete.

The system grants only those requests which are both feasible and safe. All other request are kept pending till they become both feasible and safe.

The OS uses a common sense admission criterion while initiating a process p_j , namely that the maximum no. of units of resource class k required by p_j at any time during its execution must not exceed the total no. of units of resource class k in the system . The OS may initiate any no. processes satisfying the admission criteria. Thus the total resource requirements of all admitted processes may exceed the no. of resource units of resource class k. the algorithm is called bankers algorithm presumably because bankers follow an analogous procedure while granting loans.

Safe and Unsafe States

A state (as in the above example) is considered safe if it is possible for all processes to finish executing (terminate). Since the system cannot know when a process will terminate, or how many resources it will have requested by then. The system assumes that all processes will eventually attempt to acquire their stated maximum resources and terminate soon afterward. This is a reasonable assumption in most cases since the system is not particularly concerned with how long each process runs (at least not from a deadlock avoidance perspective). Also, if a process terminates without acquiring its maximum resources, it only makes it easier on the system.

Given that assumption, the algorithm determines if a state is safe by trying to find a hypothetical set of requests by the processes that would allow each to acquire its

maximum resources and then terminate (returning its resources to the system). Any state where no such set exists is an unsafe state.

Note that these requests and acquisitions are *hypothetical*. The algorithm generates them to check the safety of the state, but no resources are actually given and no processes actually terminate. Also note that the order in which these requests are generated – if several can be fulfilled – doesn't matter, because all hypothetical requests let a process terminate, thereby increasing the system's free resources.

Requests

When the system receives a request for resources, it runs the Banker's algorithm to determine if it is safe to grant the request. The algorithm is fairly straight forward once the distinction between safe and unsafe states is understood.

1. Can the request be granted?
   if not, the request is impossible and must either be denied or put on a waiting list
2. Assume that the request is granted
3. Is the new state safe?
   If so grant the request
   If not, either deny the request or put it on a waiting list

Whether the system denies an impossible or unsafe request or makes it wait is an operating system specific decision.

Algorithm / Flowchart :