Day 1:Banker's Algorithm

Akshay Gadre

January 1^{st} 2017

1 What is it?

Banker's algorithm is a algorithm for resource allocation designed for "THE" operating system by Edward Dijkstra(against conventional wisdom, it is not designed by someone named Banker) to avoid deadlocks and check before allocating resources to a process.

The name derives from the fact that the same can be applied to the banking system where the resource would be cash and processes would be the customers.(Especially relevant in current situation of demonetization in India).

1.1 Input:

Let there be M resources and N processes. Let i denote the index of the resource and j denote the index of the process.

It takes input:

- Amount of each resource each process could possibly request M_{ij}
- Amount of each resource each process is currently holding C_{ij}
- Amount of each resource the system currently has available A_i

Resource i can be allocated to process j iff requested amount $R_{ij} \leq A_i$. For the sake of better understanding, we can create a new matrix which contains the remaining resource request of each process per resource(N_{ij}).

1.2 Output:

Whether the current state of the system is stable or not. That means whether there exists a path of sequential allocations and freeing of resources to satisfy the maximum requirement of each process and hence terminate.

2 Approach

2.1 Request:

Usually processes request a set of resources from the operating system. Whenever a request is received, the processor must find the answer to 2 questions before serving the request.

1. Does it have available resources to serve the request?
2. Does the system transition into a stable state or not after serving this request?

An allocation is done iff the answer is YES to both the questions above. The system will otherwise decide to drop, postpone or reschedule the request depending on the backoff policy of the operating system.

2.2 Algorithm:

1. See whether $R_{ij} \leq A_i$. If not, then "**backoff**".
2. Assuming the allocation is done, serially allocate and free the maximal resources of all process until all processes terminate or no process can be allocated the maximal resources.
3. If all processes terminate then "**Allocate** R_{ij}" else "**Backoff**".

The above algorithm provides a strict guarantee of deadlock-safe allocations given an initial state.

3 Limitations:

Most of the current operating systems don't know the set of maximum resources a process may request. Also the number of processes is dynamic which is difficult to handle using this algorithm. This algorithm gives a guarantee which can be relaxed to find heuristics which provide better resource utilization.