Implementation of Banker’s Algorithm

Rathna Ramesh
Arpita Saha

Problem Description

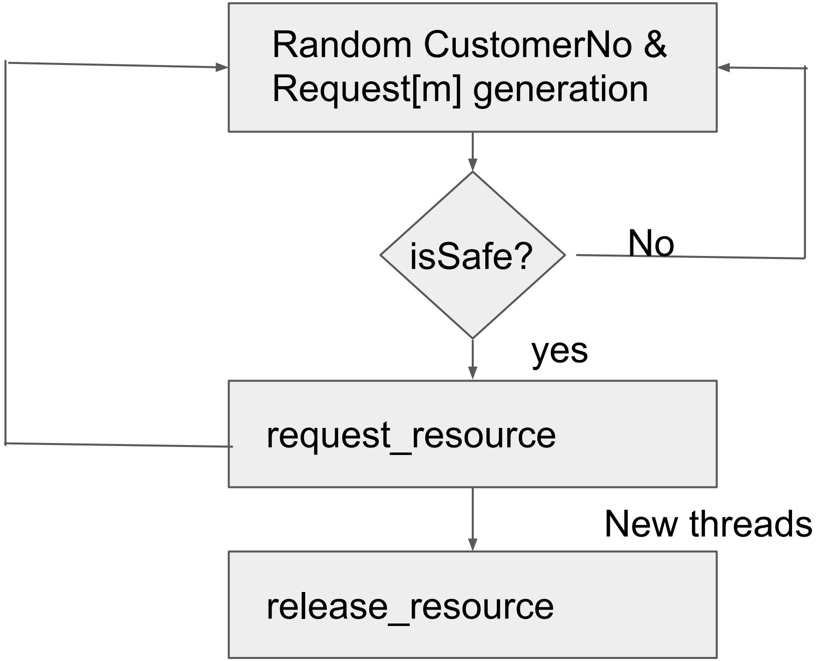
Bankers

- Considers requests from “n” customers and “m” resource types.
- Allocate to requesting customer based on availability and max limit of that customer.
- Grant a request at a time if it is safe to do so.

Customers

- “N” customer threads that request and release resources from the bank.
- Customer’s request will be limited by their need.

Code flow


Random CustomerNo & Request[m] generation
isSafe?
No
yes
request_resource
New threads
release_resource

Data Structures

1. **Available[m]** - Available[j] indicates the number of resources available of the resource j
2. **Max[n][m]** - Max[i][j] indicates the max number of resource j that can be allocated to customer i
3. **Allocation[n][m]** - Allocation[i][j] indicates the number of resources of resource j currently allocated to customer i
4. **Need[n][m]** - Need[i][j] indicates how many more of resource j that can be allocated to customer i. Need[i][j] = Max[i][j] - Allocation[i][j]

Note: n - number of customers, m - no of resources

isSafe

If the customer **i**’s request for m resources is stored in **Request[m]**, then

For every resource **j**,

Request[j] <= Need[i][j]

And Request[j] <= Available[j]

request_resource

If *isSafe* returns true,

For every resource j of the customer i,

Available[j] -= request[j];

Allocation[i][j] += request[j];

Need[i][j] -= request[j];

Request[j] number of threads are created to hold the resources for a random amount of time

release_resource

This function is called when a customer thread has reached its end. One resource of resource j will be released when this function is called.

For resource j of customer i,

```
    Available[j] +=1;

    Need[i][j] += 1;

    Allocation[i][j] -= 1;
```

End