Computer System Engineering

Lab2: Banker's Algorithm

Deadlock and Banker's algorithm

Reference: ch7_2019.pdf

What is Deadlock?

slides 7.5 – 7.16

What is Banker's algorithm?

slides 7.33 – 7.37

Tasks:

1 . Implement a basic Bank system

2 . Implement safety check algorithm

3 . Analyse the complexity of banker’s algorithm

Task 1: Bank System

1 . Customers can request resources, while system can grant the request or deny it.

2 . Customers can release resources

3 . System can print its states by invoke special function

Task 2: Safety Check Algorithm

Safety Check:
when a process requests an available resource, system must decide if granting the request will leave the system in a safe state.

The psuedocode is also available in the handout

```
boolean checkSafe (customerNumber, request) {
        temp_avail = available - request;
        temp_need(customerNumber) = need - request;
        temp_allocation(customerNumber) = allocation + request;
        work = temp_avail;
        finish(all) = false;
        possible = true;
        while(possible) {
                possible = false;
                for(customer Ci = 1:n) {
                        if(finish(Ci) == false && temp_need(Ci) <= work) {
                                possible = true;
                                work += temp_allocation(Ci);
                                finish(Ci) = true;
                        }
                }
        }
        return (finish(all) == true);
}
```

Task 3: Complexity analysis

Please analyze the time complexity of Banker's algorithm if:

1. there is n customers in total

2. There is m resources in total

Code: data structure

Java Version

```
private int numberOfCustomers;  // the number of customers
private int numberOfResources;  // the number of resources

private int[] available;    // the available amount of each resource
private int[][] maximum;    // the maximum demand of each customer
private int[][] allocation; // the amount currently allocated
private int[][] need;       // the remaining needs of each customer
```

C Version

```
int numberOfCustomers; // the number of customers
int numberOfResources; // the number of resources

int *available;       // the available amount of each resource
int **maximum;        // the maximum demand of each customer
int **allocation;     // the amount currently allocated
int **need;           // the remaining needs of each customer
```

Code: Q1-Bank System (Java)

You need to implement following functions:

```
/**
 * Constructor for the Banker class.
 * @param resources           An array of the available count for each resource.
 * @param numberOfCustomers   The number of customers.
 */
public Banker (int[] resources, int numberOfCustomers);

/**
 * Sets the maximum number of demand of each resource for a customer.
 * @param customerIndex   The customer's index (0-indexed).
 * @param maximumDemand   An array of the maximum demanded count for each resource.
 */
public void setMaximumDemand(int customerIndex, int[] maximumDemand);

/**
 * Prints the current state of the bank.
 */
public void printState();
```

Assume customerIndex will be in a valid range for simplicity.

Please refer to the starter code for more details.

Code: Q1-Bank System (Java)

You need to implement following functions:

```java
/**
 * Requests resources for a customer loan.
 * If the request leave the bank in a safe state, it is carried out.
 * @param customerIndex  The customer's index (0-indexed).
 * @param request        An array of the requested count for each resource.
 * @return true if the requested resources can be loaned, else false.
 */
public synchronized boolean requestResources(int customerIndex, int[] request);

/**
 * Checks if the request will leave the bank in a safe state.
 * @param customerIndex  The customer's index (0-indexed).
 * @param request        An array of the requested count for each resource.
 * @return true if the requested resources will leave the bank in a
 *         safe state, else false
 */
private synchronized boolean checkSafe(int customerIndex, int[] request);

/**
 * Releases resources borrowed by a customer. Assume release is valid for simplicity.
 * @param customerIndex  The customer's index (0-indexed).
 * @param release        An array of the release count for each resource.
 */
public synchronized void releaseResources(int customerIndex, int[] release);
```

Please refer to the starter code for more details.

Code: Q1-Bank System (C)

You need to implement following functions:

```
/**
 * Initializes the state of the bank.
 * @param resources  An array of the available count for each resource.
 * @param m          The number of resources.
 * @param n          The number of customers.
 */
void initBank(int *resources, int m, int n);

/**
 * Sets the maximum number of demand of each resource for a customer.
 * @param customerIndex  The customer's index (0-indexed).
 * @param maximumDemand  An array of the maximum demanded count for each resource.
 */
void setMaximumDemand(int customerIndex, int *maximumDemand);

/**
 * Prints the state of the bank.
 */
void printState();
```

Assume customerIndex will be in a valid range for simplicity.

Please refer to the starter code for more details.

Code: Q1-Bank System (C)

You need to implement following functions:

```c
/**
 * Requests resources for a customer loan.
 * If the request leave the bank in a safe state, it is carried out.
 * @param customerIndex  The customer's index (0-indexed).
 * @param request        An array of the requested count for each resource.
 * @return 1 if the requested resources can be loaned, else 0.
 */
int requestResources(int customerIndex, int *request);

/**
 * Checks if the request will leave the bank in a safe state.
 * @param customerIndex  The customer's index (0-indexed).
 * @param request        An array of the requested count for each resource.
 * @return 1 if the requested resources will leave the bank in a
 *         safe state, else 0
 */
int checkSafe(int customerIndex, int *request);

/**
 * Releases resources borrowed by a customer. Assume release is valid for simplicity.
 * @param customerIndex  The customer's index (0-indexed).
 * @param release        An array of the release count for each resource.
 */
void releaseResources(int customerIndex, int *release);
```

Please refer to the starter code for more details.

Test Cases

There are two test cases provided.
(q1.txt, q2.txt)

To help you focus on implementing the algorithm, we have provided a function in the starter code that parses and runs the test cases.

Test Case 1 (q1.txt)

n,5 — (n) Number of customers: 5

m,3 — (m) Number of resources: 3

a,10 5 7 — (a) Available resources: 10 5 7

c,0,7 5 3 — (c) Maximum resources needed by the 0th customer: 7 5 3

c,1,3 2 2

c,2,9 0 2

c,3,2 2 2

c,4,4 3 3 — (c) Maximum resources needed by the 4th customer: 4 3 3

r,0,0 1 0 — (r) Request for resources by the 0th customer: 0 1 0

r,1,2 0 0

r,2,3 0 2

r,3,2 1 1

r,4,0 0 2 — (r) Request for resources by the 4th customer: 0 0 2

f,1,2 0 0 — (f) Release of resources by the 1st customer: 2 0 0

p — (p) Print the current state of the bank

```
Customer 0 requesting
0 1 0
Customer 1 requesting
2 0 0
Customer 2 requesting
3 0 2
Customer 3 requesting
2 1 1
Customer 4 requesting
0 0 2
Customer 1 releasing
1 0 0

Current state:
Available:
4 3 2

Maximum:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3

Allocation:
0 1 0
1 0 0
3 0 2
2 1 1
0 0 2

Need:
7 4 3
2 2 2
6 0 0
0 1 1
4 3 1
```

Test Case 2 (q2.txt)

```
n,5
m,3
a,10 5 7
c,0,7 5 3
c,1,3 2 2
c,2,9 0 2
c,3,2 2 2
c,4,4 3 3
r,0,0 1 0
r,1,2 0 0
r,2,3 0 2
r,3,2 1 1
r,4,0 0 2
r,1,1 0 2
p
r,0,0 2 0
p
```

Testing the checkSafe function.

We attempt to make an loan that can leave the bank in unsafe state:

(r) Request for resources by the 0^{th} customer: 0 2 0

(p) This should print the same result as the previous print.

```
Current state:
Available:
2 3 0

Maximum:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3

Allocation:
0 1 0
3 0 2
3 0 2
2 1 1
0 0 2

Need:
7 4 3
0 2 0
6 0 0
0 1 1
4 3 1
```

Requirement

- Put your screenshots and analysis in a pdf file. Your submission should contain: one pdf report; source code.

  Please zip them up and submit before midnite, 5 Mar, 2020.

- Good luck!