Banker’s Algorithm Documentation

Chezka Gaddi

Apr 07, 2020

CONTENTS:

CHAPTER ONE

INTRODUCTION

1.1 Overview

This is a multi-threaded program that implements the Banker's Algorithm. Several customers request and release resources from the bank. The banker will grant a request only if it leaves the system in a safe state. A request that leaves the system in an unsafe state will be denied. This programming assignment combines three separate topics: (1) multi-threading, (2) preventing race conditions, and (3) deadlock avoidance.

1.2 Running the Application

The program is built by running *make* in the main project folder.

The program is invoked by passing the number of resources of each type on the command line.

```
./bankers 10 5 7
```

The *available* array is initialized with these values. Currently, the program will take in initial values for three resources.

CHAPTER

TWO

IMPLEMENTATION

2.1 Banker’s Algorithm

CHAPTER

THREE

IMPLEMENTATION

3.1 bankers.c

`void init(const char *[] argv)`

Initialize all relevant data structures and synchronization objects.

Initializes the available array to the command line inputs, the allocation array to 0, the maximum array to a random number dependent on the available array and the need array to the maximum - allocation.

The mutex lock is also initialized as well as the customer ID numbers.

Parameters

- **`argv`** – command line inputs of the number of available resources.

void **`create_customers`**`()`

Create all of the customer threads.

`int safety_test(int customer, int [] request)`

safety_test first makes a copy of the current state to run tests on. It then applies the request by subtracting the request array from the allocation array and adds the request array to the customer's allocation array. It then follows the Banker's Algorithm with the safety test.

Parameters

- **`customer`** – number of the customer making the request
- **`request`** – resources being requested

Returns 0 if safe state is found, -1 if safe state not found

void **`print_test_state`**`()`

Prints the contents of the test arrays.

void **`print_state`**`()`

Prints the current state of the system.

`int main(int argc, const char *[] argv)`

Initializes the matrices, prints the initial state of the system and syncs the customer threads.

3.2 customers.c

void ***customer_loop** (void **param*)

The customer loop first creates a request for a random number of resources dependent on their need. They send the request and keep sending that request until it is either met, or they reached the max number of request.

The loop also releases a random number resources, and releases all of of the customer's allocated resources when their need goes to 0.

Parameters

- **`param`** – The customer id

Returns 0 on success

`int request_resources(int customer_num, int [] request)`

request_resources obtains the mutex lock to stage sending the request for resources to the bank. If the request is more than what is currently available or if it exceeds the needs of the customer, the request fails.

It sends the request to the safety_test to ensure that the request is safe and sends the request through when approved.

It then prints the current state of the system and unlocks the mutex.

Parameters

- **`customer_num`** – number of requesting customer
- **`request`** – number of resources to be requested

Returns 0 if request went through, -1 if request is denied

`int release_resources(int customer_num, int [] release)`

release_resources locks the mutex and then adds the release array to the available array and subtracts the release array to the customer's allocation array.

Parameters

- **`customer_num`** – number of the customer releasing resources
- **`release`** – number of resources to be released

Returns 0 if successful, -1 if unsuccessful

int **calculate_need** (int *customer_num*)

Calculates the total number of resources that a given customer still needs.

Parameters

- **`customer_num`** – customer number to be evaluated

Returns the total number of resources the customer still needs

CHAPTER FOUR

INDICES AND TABLES

- genindex
- modindex
- search

INDEX