Ontario Tech University

SOFE 3950U / CSCI 3020I Operating Systems

LAB # 5

Banker's Algorithm

Objectives

- Gain experience using multithreaded programming in C
- Experience with locking, synchronization, and deadlock concepts
- Implement the Banker's Algorithm

Important Notes

- Work in groups of **four** students

Lab Details

Notice

It is recommended for this lab activity and others that you save/bookmark the following resources as they are very useful for C programming.

- https://en.cppreference.com/w/c
- https://www.cplusplus.com/reference/clibrary/
- https://gribblelab.org/CBootCamp/

The following resources are helpful as you will need to use signals and data structures to complete the task.

- https://www.geeksforgeeks.org/bankers-algorithm-in-operating-system-2/
- https://computing.llnl.gov/tutorials/pthreads/
- https://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html
- https://www.geeksforgeeks.org/multithreading-c-2/

Lab Activity

For this lab, you will write a multithreaded program that implements the banker's algorithm discussed in the class (see documents attached). Several customers will make requests 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 lab activity combines three separate topics:

1. **Multithreading**
2. **Preventing Race Conditions** (synchronization, mutual exclusion)
3. **Deadlock Avoidance**

Clarifications

The Banker

The banker will consider requests from ***n*** customers for ***m*** resources types. As outlined in **Section 7.5.3 from the additional file**. The banker will keep track of the resources using the following data structures, however feel free to **modify or write your own data structures if necessary**.

```
// May be any values >= 0
#define NUM_CUSTOMERS 5
#define NUM_RESOURCES 3
// Available amount of each resource
int available[NUM_RESOURCES];
// Maximum demand of each customer
int maximum[NUM_CUSTOMERS][NUM_RESOURCES];
// Amount currently allocated to each customer
int allocation[NUM_CUSTOMERS][NUM_RESOURCES];
// Remaining need of each customer
int need[NUM_CUSTOMERS][NUM_RESOURCES];
```

The Customers

Create ***n*** customer threads that request and release resources from the **bank**. The customers will continually loop, requesting and then releasing random numbers of resources. The customers' requests for resources will be bounded by their respective values in the need array. The banker will grant a request if it satisfies the safety algorithm outlined in the **document attached**. If a request does not leave the system in a safe state, the banker will deny it. **Function prototypes for requesting and releasing resources** are as follows:

```
bool request_res(int n_customer, int request[]);
bool release_res(int n_customer, int release[]);
```

These two functions should **return true if successful** (the request has been granted) and **false if unsuccessful**. Multiple threads (customers) will **concurrently access shared data through these two functions**. Therefore, access must be controlled through **mutex locks to prevent race conditions**. Implement the mutex locks using the pthread library as discussed in the previous lectures, tutorials and labs.

Implementation

Your program is executed by passing the number of resources of each type on the command line. For example, if there were three resource types (#define NUM_RESOURCES 3), with **ten** instances of the first type, **five** of the second type, and **seven** of the third type, you would invoke your program follows:

./banker 10 5 7

The available array would be initialized to these three values. The **maximum** array (which holds the maximum demand of each customer) can be initialized using any method that you find convenient. Your program **must** print the output of the banker each time the resources are requested and released, and should contain the following information:

- The current thread making the request
- The resources requested
- If the system will be in a safe or unsafe state
- The outcome (accepted/denied), if the resources are granted or not
- Resources released, if there are any resources released

Deliverables

1. All source files for your implementation, include a makefile so that the code can be compiled. The solution **MUST** make use of threading in order to receive marks.

2. Your program **MUST** take the resources as command line arguments as specified in the implementation details. You **MUST** print the output of the Banker's Algorithm as outlined in the implementation details section, feel free to add additional output if it will enhance the demonstration of your implementation of the Banker's Algorithm.