CP386: Assignment 4 – Spring 2020

Due on July 30, 2020

Before 11:59 PM

In this assignment we will try to practice the concept of deadlock avoidance.

General Assignment Notes

For collaborating and managing the source code on GitHub

- You have to create a public repository on GitHub, where you and your teammate would collaborate on this code development.
- You have to maintain the README.md file for the repository to explain the following:
    - Project title
    - Motivation
    - Installation
    - Screenshots
    - Individual contribution (Function-wise)
    - Features
    - Tests
    - Code Example
    - Authors
    - Credits
    - License
- Marks for you and your teammate's contribution to the project would be allotted based on the history of the commits to the GitHub. If no commit is seen from your login ID then zero mark would be awarded to you.

Even if you are creating the GitHub project you have to submit the source code file to Myls page. For submitting your assignments to Myls page follow these requirements:

- Name your source code file as: combination of your Laurier ID number & your teammate's ID, an underscore, 'a' (for 'assignment'), then the assignment number in two digits. For example, if the user 100131001 and 100131233 submits Assignment 4, the name should be: `100131001_100131233_a04.c.txt`. No other file name format will be accepted. We require the .txt extension in the end because MyLS does not allow .c extension.
- Include in source code file: your and your teammate's GitHub login ID and the URL of the public GitHub repository used for your project/algorithm development.
- For this assignment you must use C99 language syntax. Your code must be compiled using make **without errors.** There will be a small bonus if it compiles without warnings. You have to create with a make file and mention instructions to use it on GitHub page.
- **Test your program thoroughly with the `gcc` compiler (version 5.4.0) in a Linux environment.**

- If your code does not compile, **then you will get zero**. Therefore, make sure that you have removed all syntax errors from your code.

Note:

1. *GitHub repository is must for this assignment. If you are not maintaining it then you will score zero in Assignment 4. Further, marks will be deducted from any questions where these requirements are not met.*
2. *This is a group assignment and for creating groups instruction would be provided on Myls.*

Description

In this assignment, you will write a multiple threaded program that implements the banker's algorithm. Customers request and release resources from the bank. The banker will keep track of the resources. The banker will grant a request if it satisfies the safety algorithm. If a request does not leave the system in a safe state, the banker will deny it.

- Your program should consider requests from n customers for m resources types. The banker will keep track of the resources using the following data structures as mentioned in chapter 8 of textbook:
  - the `available` amount of each resource
  - the `maximum` demand of each customer
  - the amount currently `allocated` to each customer
  - the remaining `need` of each customer
- It implements the function for requesting resource, it should return 0 if successful and −1 if unsuccessful.
- It implements the function for releasing resources.
- The program includes a safety algorithm to grant a request, if it does leave the system in a safe state, otherwise will deny it.
- The program allows the user to interactively enter a request for resources, to release resources, or to output the values of the different data structures (`available`, `maximum`, `allocation`, and `need`) used with the banker's algorithm.
- The program should be invoked by passing the number of resources of each type via command line to intialisze the `available` array by these values. For example, if there were four resource types, with ten instances of the first type, five of the second type, seven of the third type, and eight of the fourth type, you would invoke your program as follows:

```
./assignment04.out 10 5 7 8
```

- Read the sample input file, “sample4_in.txt” containing the maximum number of requests for each customer (five customers and four resources). where each line in the input file represents the maximum request of each resource type for each customer. Your program will initialize the `maximum` array to these values.
- Program would take user enter commands for responding to a request of resources, a release of resources, or the current values of the different data structures. Use the command ‘RQ’ for requesting resources (remember threads cannot request more than maximum number of resource for that thread), ‘RL’ for releasing resources, and ‘*’ to output the values of the different data structures, ‘Run’ find the safe sequence and run each thread.

- For example, if customer/thread 0 were to request the resources (3, 1, 2, 1), the following command would be entered:

```
RQ 0 3 1 2 1
```

- The 'RQ' command would fill the `allocation` array. Program would then output whether the request would be satisfied or denied using the safety algorithm outlined in chapter 8 of textbook. The user would use 'RQ' to request for the resources for all the customers/threads. The example interaction would be:

```
osc@ubuntu:~/A04/bankers-algorithm$ ./a.out 10 5 7 8
Number of Customers: 5
Currently Available resources: 10 5 7 8
Maximum resources from file:
6,4,7,3
4,2,3,2
2,5,3,3
6,3,3,2
5,6,7,5
Enter Command: RQ 0 3 1 2 1
Request is satisfied
Enter Command: RQ 1 1 1 1 1 (Similarly, enter requests for other
customers/threads)
```

- Similarly, if customer 4 were to release the resources (1, 2, 3, 1), the user would enter the following command:

```
RL 4 1 2 3 1
```

- The command '*' is entered, your program would output the current state of the `available, maximum, allocation`, and `need` arrays.
- The command 'Run' would execute the safe sequence based on the current state and all the threads would be run same function code and prints (for example, but not restricted to):

```
Safe Sequence is:  <0 1 2 4 3>
Now going to executing the threads:


--> Customer/Thread 0
        Allocated resources:   3 1 2 1
        Needed:   3 3 5 2
        Available:   4 3 6 3
        Thread has started
        Thread has finished
        Thread is releasing resources
        New Available: 7 4 8 4
--> Customer/Thread 1 (Similarly, print individual information for other
customers/threads)
```

- The other implementation details are on your discretion and you are free to explore.
- You have to write all the functions (including safety algorithm) those are required to implement the Banker's algorithm. Complete the program as per following details so that we can have functionality as described above. Write all the code in single C file.