CPSC 351: OS - Fall 2020

Project One, **Banker's Algorithm,** due Friday, 6 Nov 2020

In this assignment, you will create a solution for the banker's algorithm project from section 8.6.3 (Banker's Algorithm) in the 10th edition of Silberschatz (Operating System Concepts), or the 8th or 9th ed. of Stallings (Operating Systems: Internals and Design Principles). Your solution can be in either: **C, C++, or Java**. A partial, possible layout of your project in C++ and in Java is attached to the project description in Slack #general and in Titanium. The description of this algorithm is in ch. 8 of Silberschatz, and in ch. 6 of Stallings.

Your simulation should contain the essentials of the Banker's Algorithm (see rubric at end of this assignment description), namely, customers (threads) should make requests of the banker for a limited set of resources. The banker should check whether the customer is exceeding the maximum number of resources he/she said it would ever request, and if there are sufficient system resources, and if the request would put the system in an unsafe state.

To determine if the system would be in an unsafe state, the algorithm should attempt to grant the request, by increasing the resources allocated to the given process, and by reducing the total resources still needed by it to satisfy the maximum resources it needs. If there is a still a path of allocating resources and having them released that will satisfy all of the processes that are running, then the system is in a safe state, and the request should be approved.

Only safe requests should be approved.
Once the request is approved, the state of the system (Allocated, Need matrices and Available vector) should be updated.

You can either have the customer make requests until they reach their maximum needed, then release all resources and exit. Here is an example of the output from your program.

BANKER'S PROJECT example OUTPUT Factory
10 5 7 adding customer 0... adding customer
1... adding customer 2... adding customer 3...
adding customer 4... FACTORY: created
threads
FACTORY: started threads
#P0 RQ:[3 1 1],needs:[7 4 3],available=[10 5 7] ---> APPROVED, #P0 now at:[3 2 1] available [7 4 6]

	ALLOCATED	MAXIMUM	NEED
	[3 2 1]	[7 5 3]	[4 3 2]
[2	0 0]	[3 2 2]	[1 2 2]
[3	0 2]	[9 0 2]	[6 0 0]
	[2 1 1]	[2 2 2]	[0 1 1]
	[0 0 2]	[4 3 3]	[4 3 1]

#P4 RQ:[4 2 1],needs:[4 3 1],available=[7 4 6] ---> APPROVED, #P4 now at:[4 2 3] available [3 2 5]

	ALLOCATED	MAXIMUM	NEED
	[3 2 1]	[7 5 3]	[4 3 2]
[2	0 0]	[3 2 2]	[1 2 2]
[3	0 2]	[9 0 2]	[6 0 0]
	[2 1 1]	[2 2 2]	[0 1 1]
	[4 2 3]	[4 3 3]	[0 1 0]

#P3 RQ:[0 1 0],needs:[0 1 1],available=[3 2 5] ---> APPROVED, #P3 now at:[2 2 1] available [3 1 5]

	ALLOCATED	MAXIMUM	NEED
	[3 2 1]	[7 5 3]	[4 3 2]
[2	0 0]	[3 2 2]	[1 2 2]
[3	0 2]	[9 0 2]	[6 0 0]
	[2 2 1]	[2 2 2]	[0 0 1]
	[4 2 3]	[4 3 3]	[0 1 0]

#P0 RQ:[1 2 0],needs:[4 3 2],available=[3 1 5] DENIED
#P2 RQ:[2 0 0],needs:[6 0 0],available=[3 1 5] ---> APPROVED, #P2 now at:[5 0 2] available [1 1 5]

ALLOCATED	MAXIMUM	NEED
[3 2 1]	[7 5 3]	[4 3 2]
[2 0 0]	[3 2 2]	[1 2 2]
[5 0 2]	[9 0 2]	[4 0 0]

```
[2 2 1]	[2 2 2]	[0 0 1]
[4 2 3]	[4 3 3]	[0 1 0]
#P4 RQ:[0 1 0],needs:[0 1 0],available=[1 1 5]   ---> APPROVED, #P4 now at:[4 3 3] available [1 0 5]
	ALLOCATED	MAXIMUM	NEED
[3 2 1]	[7 5 3]	[4 3 2]
[2 0 0]	[3 2 2]	[1 2 2]
[5 0 2]	[9 0 2]	[4 0 0]
[2 2 1]	[2 2 2]	[0 0 1]
[4 3 3]	[4 3 3]	[0 0 0]
#P1 RQ:[0 1 1],needs:[1 2 2],available=[1 0 5] DENIED
#P2 RQ:[4 0 0],needs:[4 0 0],available=[1 0 5] DENIED
#P2 RQ:[2 0 0],needs:[4 0 0],available=[1 0 5] DENIED
#P2 RQ:[3 0 0],needs:[4 0 0],available=[1 0 5] DENIED
#P0 RQ:[1 3 2],needs:[4 3 2],available=[1 0 5] DENIED
#P0 RQ:[3 3 2],needs:[4 3 2],available=[1 0 5] DENIED
#P3 RQ:[0 0 1],needs:[0 0 1],available=[1 0 5]   ---> APPROVED, #P3 now at:[2 2 2] available [1 0 4]
	ALLOCATED	MAXIMUM	NEED
[3 2 1]	[7 5 3]	[4 3 2]
[2 0 0]	[3 2 2]	[1 2 2]
[5 0 2]	[9 0 2]	[4 0 0]
[2 2 2]	[2 2 2]	[0 0 0]
[4 3 3]	[4 3 3]	[0 0 0]
#P0 RQ:[1 0 1],needs:[4 3 2],available=[1 0 4]   ---> APPROVED, #P0 now at:[4 2 2] available [0 0 3]
	ALLOCATED	MAXIMUM	NEED
[4 2 2]	[7 5 3]	[3 3 1]
[2 0 0]	[3 2 2]	[1 2 2]
[5 0 2]	[9 0 2]	[4 0 0]
[2 2 2]	[2 2 2]	[0 0 0]
[4 3 3]	[4 3 3]	[0 0 0]
#P1 RQ:[0 2 1],needs:[1 2 2],available=[0 0 3] DENIED
#P1 RQ:[0 2 2],needs:[1 2 2],available=[0 0 3] DENIED
#P2 RQ:[2 0 0],needs:[4 0 0],available=[0 0 3] DENIED
------------------------------> #P3 has all its resources!  RELEASING ALL and SHUTTING DOWN...
======================== customer #3 releasing:[2 2 2],allocated=[0 0 0]
------------------------------> #P4 has all its resources!  RELEASING ALL and SHUTTING DOWN...
======================== customer #4 releasing:[4 3 3],allocated=[0 0 0] #P1 RQ:[0 1 2],needs:[1 2 2],available=[6 5 8]
---> APPROVED, #P1 now at:[2 1 2] available [6 4 6]
	ALLOCATED	MAXIMUM	NEED
[4 2 2]	[7 5 3]	[3 3 1]
[2 1 2]	[3 2 2]	[1 1 0]
[5 0 2]	[9 0 2]	[4 0 0]
-------	-------	-------
-------	-------	-------
#P2 RQ:[3 0 0],needs:[4 0 0],available=[6 4 6]   ---> APPROVED, #P2 now at:[8 0 2] available [3 4 6]
	ALLOCATED	MAXIMUM	NEED
[4 2 2]	[7 5 3]	[3 3 1]
[2 1 2]	[3 2 2]	[1 1 0]
[8 0 2]	[9 0 2]	[1 0 0]
-------	-------	-------
-------	-------	-------
#P0 RQ:[1 0 0],needs:[3 3 1],available=[3 4 6]   ---> APPROVED, #P0 now at:[5 2 2] available [2 4 6]
	ALLOCATED	MAXIMUM	NEED
[5 2 2]	[7 5 3]	[2 3 1]
[2 1 2]	[3 2 2]	[1 1 0]
[8 0 2]	[9 0 2]	[1 0 0]
-------	-------	-------
-------	-------	-------
#P1 RQ:[0 1 0],needs:[1 1 0],available=[2 4 6]   ---> APPROVED, #P1 now at:[2 2 2] available [2 3 6]
	ALLOCATED	MAXIMUM	NEED
[5 2 2]	[7 5 3]	[2 3 1]
[2 2 2]	[3 2 2]	[1 0 0]
[8 0 2]	[9 0 2]	[1 0 0]
-------	-------	-------
-------	-------	-------
#P2 RQ:[1 0 0],needs:[1 0 0],available=[2 3 6]   ---> APPROVED, #P2 now at:[9 0 2] available [1 3 6]
	ALLOCATED	MAXIMUM	NEED
[5 2 2]	[7 5 3]	[2 3 1]
```

```
            [2 2 2]     [3 2 2]     [1 0 0]
            [9 0 2]     [9 0 2]     [0 0 0]
            -------     -------     -------
            -------     -------     -------
#P0 RQ:[0 2 0],needs:[2 3 1],available=[1 3 6]   ---> APPROVED, #P0 now at:[5 4 2] available [1 1 6]
            ALLOCATED   MAXIMUM     NEED
            [5 4 2]     [7 5 3]     [2 1 1]
            [2 2 2]     [3 2 2]     [1 0 0]
            [9 0 2]     [9 0 2]     [0 0 0]
            -------     -------     -------
            -------     -------     -------
#P1 RQ:[1 0 0],needs:[1 0 0],available=[1 1 6]   ---> APPROVED, #P1 now at:[3 2 2] available [0 1 6]
            ALLOCATED   MAXIMUM     NEED
            [5 4 2]     [7 5 3]     [2 1 1]
            [3 2 2]     [3 2 2]     [0 0 0]
            [9 0 2]     [9 0 2]     [0 0 0]
            -------     -------     -------
            -------     -------     -------
------------------------------> #P2 has all its resources!  RELEASING ALL and SHUTTING DOWN...
======================== customer #2 releasing:[9 0 2],allocated=[0 0 0]
------------------------------> #P1 has all its resources!  RELEASING ALL and SHUTTING DOWN...
======================== customer #1 releasing:[3 2 2],allocated=[0 0 0]
#P0 RQ:[1 0 1],needs:[2 1 1],available=[12 3 10]        ---> APPROVED, #P0 now at:[6 4 3] available [11 3 9]
            ALLOCATED   MAXIMUM     NEED
            [6 4 3]     [7 5 3]     [1 1 0]
            -------     -------     -------
            -------     -------     -------
            -------     -------     -------
            -------     -------     -------
#P0 RQ:[0 1 0],needs:[1 1 0],available=[11 3 9]  ---> APPROVED, #P0 now at:[6 5 3] available [11 2 9]
            ALLOCATED   MAXIMUM     NEED
[6  5 3]    [7 5 3]     [1 0 0]
            -------     -------     -------
            -------     -------     -------
            -------     -------     -------
            -------     -------     -------
#P0 RQ:[1 0 0],needs:[1 0 0],available=[11 2 9]  ---> APPROVED, #P0 now at:[7 5 3] available [10 2 9]
            ALLOCATED   MAXIMUM     NEED
[7  5 3]    [7 5 3]     [0 0 0]
            -------     -------     -------
            -------     -------     -------
            -------     -------     -------
            -------     -------     -------
------------------------------> #P0 has all its resources!  RELEASING ALL and SHUTTING DOWN...
======================== customer #0 releasing:[7 5 3],allocated=[0 0 0] Process finished with exit code 0
```

Submission

Turn in the code for this project by uploading all of the source files you created to a single public repository on GitHub. While you may discuss this homework assignment with other students, work you submit must have been completed on your own.

To complete your submission, print the sheet at the back of this file, fill out its spaces, and submit it to the instructor in class by the deadline. Failure to follow the instructions exactly will incur a 10% penalty on the grade for this assignment. Please submit your assignment to your github repository as a SINGLE ZIP FILE with the following format: last name_firstname_bankers.zip, ALL IN LOWER CASE.

CPSC 351 Project 1 – Banker's Algorithm, due Friday, 6 Nov 2020

Your name: Bryan Monh

Repository *(print)***:** https://github.com/monhbryan/banker

Verify each of the following items and place a checkmark in the correct column. Each item incorrectly marked will incur a 5% penalty on the grade for this assignment

Finished	Not finished	
✓	□	Created the Banker's Algorithm in C, C++, or Java, so that customers make requests of the banker for a limited set of resources.
✓	□	t does not allow requests to be granted that would result in the system being in an unsafe state., by granting a proposed request and trying to find a path that satisfies all processes.
✓	□	Reads from and uses multiple input text files to run the simulation. Each text file contains the resources for the Bank's Available resources on its first line, and the Allocated and Maximum resources for each Customer on its subsequent lines.
✓	□	Uses a Vector-like class, such as Vector<E> (Java), std::vector<T> (C++), or a subclass that allows vectors to be added, subtracted, and compared.
✓	□	It uses a Bank class to simulate the Bank, which has a vector (Avail) that holds a set of resources that will be requested by Customers. The Bank determines if it is safe to allocate the resources to the customer, and approves the request or denies it.
✓	□	It uses a Customer class, which has an Alloc vector to hold its resources, a Max vector to indicate the maximum number of resources it requires, and a Needs vector to indicate how many resources it needs to reach its maximum.
✓	□	Uses threads to simulate customers making requests and releasing resources when their maximum needs are satisfied
✓	□	Has Customers make random requests from the Bank, and has the Bank determine if the request is safe using the Banker's algorithm. Namely, it temporarily grants the request, then determines if there is still a path to satisfy all other Customers to reach their Max resources required. If there is, the request is approved; if not, it is denied.
✓	□	Each time a request is approved, the Bank prints out the entire state of the system: its Avail vector, and each Customer's Alloc, Max, and Need vectors.
✓		Once a Customer reaches its Max resources, it releases them all and shuts down its thread. The simulation runs until all Customers have shut down, or runs until it is in a deadlock it cannot recover from (e.g., the system's algorithm either fails, or the system starts in deadlock).
✓	□	Tested the simulation with different input files that are known to have the system in a safe state to see if the simulation can find the solution (use Silberschatz and Stallings).
✓	□	Has a S**imulation state**, that has processes make random requests for resources, and attempts to find a safe path for all processes to run to completion and release their resources. The simulation ends when all processes have shut down. (see examples)
✓	□	Project directory pushed to new GitHub repository listed above using GitHub client.

Comments: Note that I use the provided examples and modified the examples directly to implement the algorithm

Fill out and print this page, and submit it on the day this project is due.