CP386: Assignment 5 – Fall 2022

Due on Dec 05, 2022

Before 11:59 PM

It is a group (of two) assignment to practice the concept of multiple resource allocation and deadlock avoidance related programming concepts.

General Instructions:

- For this assignment, you must use C language syntax. Your code must compile using make **without errors.** You will be provided with a Makefile and instructions on using it.
- **Test your program thoroughly with the GCC compiler in a Linux environment.**
- If your code does not compile, **then you will score zero**. Therefore, ensure you have removed all syntax errors from your code.
- **Gradescope** platform will be used to upload the assignment file(s) for grading. The link to the Gradescope assignment is available on Myls course page. For submission, Drag and drop your code file(s) into Gradescope. **Make sure your file name is as suggested in the assignment; using a different name may score Zero.**
- Please note that the submitted code will be checked for plagiarism. By submitting the code file(s), you would confirm that you have not received unauthorized assistance in preparing the assignment. You also confirm that you are aware of course policies for submitted work.
- Marks will be deducted for any questions where these requirements are not met.
- Multiple attempts will be allowed, but only your last submission before the deadline will be graded. We reserve the right to take off points for not following directions.

Question 1 (Marks: 50):

In this question, you will write a multi-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` resource 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
- The program includes a safety algorithm to grant a request, if it leaves 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 and executes customers as threads in a safe sequence.

- The program should be invoked by passing the number of available resources of each type via command line to initialize the `available` array by these values. For example, if in system 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:

```
./banker 10 5 7 8
```

- The program should read the sample input file, "sample_in_bankers.txt" containing the maximum number of requests for each customer (e.g., 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.
- The program would run a loop (user enters `Exit` to stop its execution) and would take user enter commands for responding to request of resources, release of resources, the current values of the different data structures, or find the safe sequence and run each thread. The program should take the following commands:
  - `<RQ [int customer_number] [int Resource 1] [int Resource 2] ... [int Resource 4]>`: `'RQ'` command is used for requesting resources (remember threads cannot request more than maximum number of resources for that thread). If the request leaves the system unsafe it will be denied. If the system state is safe, the resources would be allocated and a message `"State is safe, and request is satisfied"` would be printed.
  - `<RL [int customer_number] [int Resource 1] [int Resource 2] ... [int Resource 4]>`: 'RL' command would release the resources and data structures would be updated accordingly. It would print `"The resources have been released successfully"`.
  - `<Status>`: `'Status'` command would print all arrays and matrices used (`available`, `maximum`, `allocation`, and `need`).
  - `<Run>`: `'Run'` command executes customers as threads in a safe sequence. Each thread requests the resources it needs, releases them, and lets the next thread in the sequence run.
  - `<Exit>`: The `'Exit'` command is used to exit the loop and the program
- For example, if customer/thread 0 were to request the resources (1, 0, 0, 1), the following command would be entered:

```
RQ 0 1 0 0 1
```

- The `'RQ'` command would be used to fill the `allocation` array. The `'RQ'` command, calls a safety algorithm. Then it outputs whether the request would be satisfied or denied using the safety algorithm outlined in chapter 8 of textbook. The example interaction would be:

```
osc@ubuntu:~/A04/bankers-algorithm$ ./banker 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 5 7 5
Enter Command: RQ 0 1 0 0 1
State is safe, and request is satisfied
Enter Command: RQ 1 1 1 1 1
State is safe, and request is satisfied
```

```
Enter Command:
```

- For example, if customer 4 wishes to release the resources (1, 0, 0, 0), the user would enter the following command:

```
RL 4 1 0 0 0
```

- When the command `'Status'` 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 at any time (if this command `'Run'` is entered) and all the customers (threads) would be run same function code and prints for each thread (but not restricted to):
    - The safe sequence
    - Name of thread running in sequence ordering
    - Allocated Resources
    - Need
    - Available resources
    - A message: `"Thread has started"`
    - A message: `"Thread has finished"`
    - A message: `"Thread is releasing resources"`
    - New Available status.
- After `'Run'`, you can again ask user to enter a new command and make allocations till user does not enter `'Exit'` command.
- If user enters a wrong command, then a message, `Invalid input, use one of RQ, RL, Status, Run, Exit,` must be displayed and the correct command must be asked.
- The other implementation details are on your discretion, and you are free to explore.
- You must write all the functions (including safety algorithm) required to implement the Banker's algorithm. Complete the program as per following details so that you can have functionality as described above. Write the complete code in a single C file.

The expected output is as follows:

```
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 5 7 5
Enter Command: RQ 0 1 0 0 1
State is safe, and request is satisfied
Enter Command: RQ 1 1 1 1 1
State is safe, and request is satisfied
Enter Command: RQ 2 2 2 2 2
State is safe, and request is satisfied
Enter Command: RQ 3 1 1 1 1
State is safe, and request is satisfied
Enter Command: RQ 4 1 0 0 0
```

```
State is safe, and request is satisfied
Enter Command: Status
Available Resources:
4 1 3 3
Maximum Resources:
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 5 7 5
Allocated Resources:
1 0 0 1
1 1 1 1
2 2 2 2
1 1 1 1
1 0 0 0
Need Resources:
5 4 7 2
3 1 2 1
0 3 1 1
5 2 2 1
4 5 7 5
Enter Command: Run
Safe Sequence is: 1 3 2 4 0
--> Customer/Thread 1
    Allocated resources:  1 1 1 1
    Needed: 3 1 2 1
    Available:  4 1 3 3
    Thread has started
    Thread has finished
    Thread is releasing resources
    New Available:  5 2 4 4
--> Customer/Thread 3
    Allocated resources:  1 1 1 1
    Needed: 5 2 2 1
    Available:  5 2 4 4
    Thread has started
    Thread has finished
    Thread is releasing resources
    New Available:  6 3 5 5
--> Customer/Thread 2
    Allocated resources:  2 2 2 2
    Needed: 0 3 1 1
    Available:  6 3 5 5
    Thread has started
    Thread has finished
    Thread is releasing resources
    New Available:  8 5 7 7
--> Customer/Thread 4
    Allocated resources:  1 0 0 0
    Needed: 4 5 7 5
    Available:  8 5 7 7
    Thread has started
    Thread has finished
    Thread is releasing resources
```

```
    New Available:  9 5 7 7
--> Customer/Thread 0
    Allocated resources:  1 0 0 1
    Needed: 5 4 7 2
    Available:  9 5 7 7
    Thread has started
    Thread has finished
    Thread is releasing resources
    New Available:  10 5 7 8
Enter Command:
```

Note: When submitting source code file for this question, name it like:

- `banker.c`