CP386: Assignment 4 – Winter 2022

Due on April 2, 2022

Before 11:59 PM

This is a group assignment (group of 2) and a GitHub repository would be used to manage the code development. In this assignment, we will try to practice the concept of deadlock avoidance and contiguous memory allocation.

General Assignment Notes:

For collaborating and managing the source code on GitHub

- You must create a repository on GitHub (***make it public only after assignment submission deadline passed***), where you and your teammate would collaborate on this code development.
- You must maintain the README.md file for the repository to explain the following:

    - Project title
    - Makefile
    - Screenshots
    - Individual contribution (Function-wise)
    - Features
    - Tests cases (if any)
    - Use examples
    - About Developers
    - License to use your code

- 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 found from your login ID, then a zero would be awarded to you.

Even if you are creating the GitHub project, you must 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 (e.g., for question 1) `100131001_100131233_a04_q1.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 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 must 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 score a 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 a zero in Assignment 4. Further, marks will be deducted for any of the questions in which these requirements are not fulfilled.*
2. *This is a group assignment and for creating groups, instruction would be provided on Myls.*

Question 1 (Marks: 40):

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:

```
./Question1 10 5 7 8
```

- The program should read the sample input file, "sample4_in.txt" (can be hard coded in the program) 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$ ./Question1 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 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:
```

Question 2 (Marks: 25):

In this program, you would be using Best-Fit algorithm for contiguous memory allocation (refer to section 9.2 from text). This project will involve managing a contiguous region of memory of size MAX where addresses may range from 0 … MAX − 1. Your program must respond to three different requests:

- Request for a contiguous block of memory
- Release of a contiguous block of memory
- Report the regions of free and allocated memory

- Your program will allocate memory using Best-Fit algorithm. This will require that your program keep track of the different allocations and holes representing available memory. When a request for memory arrives, it will allocate the memory from one of the available holes based on the allocation strategy (for the first allocation, all memory is available). If there is insufficient memory to allocate to a request, it will output an error message and reject the request
- Your program will also keep track of which region of memory has been allocated to which process. This is necessary to support the `` `Status' `` command and is also needed when memory is released via the `` `RL' `` command, as the process releasing memory is passed to this command. If a partition being released is adjacent to an existing hole, be sure to combine the two holes into a single hole.
- The program should be invoked by passing the size initial amount of the memory via command line. You would invoke your program as follows:

```
./Question2 1000000
```

- The program would run a loop (user enters Exit to stop its execution) and would take user enter commands for responding to request of contiguous memory block allocation, release of the contiguous block of memory, status/report of the regions of free and allocated memory blocks. The program should take the following commands:
    - RQ <process number> <size> <B>: `'RQ'` command is the new process that requires the memory, followed by the amount of memory being requested, and finally the algorithm. (In this program, "B" refers to best fit.)
    - RL <process number/name>: `'RL'` command will release the memory that has been allocated to a process (e.g., P0).
    - Status: The `'Status'` command used for reporting the status of memory is entered.
    - Exit: The `'Exit'` command is used to exit the loop and the program.
- A request for 20,000 bytes will appear as follows:

```
command>RQ P0 20000 B
```

  The first parameter to the RQ command is the new process that requires the memory, followed by the amount of memory being requested, and finally the strategy. (In this program, "B" refers to best fit algorithm.)
- Similarly, a release will appear as:

```
command>RL P0
```

- The other implementation details are on your discretion, and you are free to explore.

- You must write all the functions required to implement the contiguous memory allocation algorithm. Complete the program as per following details so that you can have functionality as described above. Write all the code in single C file.

The expected output is given as below:

```
$ ./Question2 1000000
Allocated 1000000 bytes of memory
command>RQ P0 200000 B
Successfully allocated 200000 to process P0
command>RQ P1 350000 B
Successfully allocated 350000 to process P1
command>RQ P2 300000 B
Successfully allocated 300000 to process P2
command>RL P0
releasing memory for process P0
Successfully released memory for process P0
command>Status
Partitions [Allocated memory = 650000]:
Address [200000:549999] Process P1
Address [550000:849999] Process P2

Holes [Free memory = 350000]:
Address [0:199999] len = 200000
Address [850000:999999] len = 150000
command>RQ P3 120000 B
Successfully allocated 120000 to process P3
command>Status
Partitions [Allocated memory = 770000]:
Address [200000:549999] Process P1
Address [550000:849999] Process P2
Address [850000:969999] Process P3

Holes [Free memory = 230000]:
Address [0:199999] len = 200000
Address [970000:999999] len = 30000
command>RQ P4 150000 B
Successfully allocated 150000 to process P4
command>RQ P5 80000 B
No hole of sufficient size
command>Status
Partitions [Allocated memory = 920000]:
Address [0:149999] Process P4
Address [200000:549999] Process P1
Address [550000:849999] Process P2
Address [850000:969999] Process P3

Holes [Free memory = 80000]:
Address [150000:199999] len = 50000
Address [970000:999999] len = 30000
```