CP386: Assignment 5 – Fall 2024

Due on December 2, 2024

Before 11:59 PM

It is a group (of two) assignment to practice 1) multiple resource allocation and deadlock avoidance-related programming concepts and 2) contiguous memory allocation.

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.
- GitHub Classroom-based repository will be used to keep daily track of your work and its split with your partner. Make sure to follow the steps described at the end of the document to create your repository.
- 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, connect GitHub Classroom repository and select the branch master to submit on Gradescope. Make sure your file names are 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`

Question 2 (Marks: 50):

- In this program, you would be developing any one of the different algorithms (first fit, best fit, worst fit) 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 any one of the first fit, best fit, and worst 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 of 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 the command line. You would invoke your program as follows:

```
./allocation 1000000
```

- The program would run a loop (user enters Exit to stop its execution) and would take the user to enter commands for responding to a request of contiguous memory block allocation, the 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. Use the following flags for different algorithm allocation approaches (remember you are supposed to develop at least one approach):
  - "F" for the first fit algorithm
  - "B" for best fit algorithm
  - "W" for worst fit algorithm

  Note: You must mention in the approach that you have implemented.
- RL <process number/name>: `'RL'` command will release the memory that has been allocated to a process (e.g., P0).
- C: The `` `C' `` command is used to compact the set of holes into one larger hole. For example, if you have four separate holes of size 550 KB, 375 KB, 1,900 KB, and 4,500 KB, your program will combine these four holes into one large hole of size 7,325 KB. There are several strategies for implementing compaction, one of which is suggested in Section 9.2.3. Be sure to update the beginning address of any processes that have been affected by compaction.
- 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 example, "B" refers to the best-fit algorithm.)
- Similarly, a release will appear as:

  ```
  command>RL P0
  ```

- The other implementation details are at your discretion, and you are free to explore.
- You must write all the functions required to implement the contiguous memory allocation algorithms. Complete the program as per the following details so that you can have functionality as described above. Write all the code in a single C file.
- To run the code run command <make run> that would initialize the program with 1 MB (1,048,576 bytes) of memory.

The expected output is given as below:

```
$ make runq2
gcc -std=gnu99 -o allocation allocation.c
./allocation 1048576
Here, the Best Fit approach has been implemented and the allocated 1048576
bytes of memory.
allocator>RQ P0 200000 B
Successfully allocated 200000 to process P0
allocator>RQ P1 350000 B
Successfully allocated 350000 to process P1
allocator>RQ P2 300000 B
Successfully allocated 300000 to process P2
allocator>RL P0
```

```
releasing memory for process P0
Successfully released memory for process P0
allocator>Status
Partitions [Allocated memory = 650000]:
Address [200000:549999] Process P1
Address [550000:849999] Process P2

Holes [Free memory = 398576]:
Address [0:199999] len = 200000
Address [850000:1048575] len = 198576
allocator>RQ P3 120000 B
index = 0 delta = 80000 best delta = 1048577
index = 1 delta = 78576 best delta = 80000
Successfully allocated 120000 to process P3
allocator>Status
Partitions [Allocated memory = 770000]:
Address [200000:549999] Process P1
Address [550000:849999] Process P2
Address [850000:969999] Process P3

Holes [Free memory = 278576]:
Address [0:199999] len = 200000
Address [970000:1048575] len = 78576
allocator>RQ P4 150000 B
index = 0 delta = 50000 best delta = 1048577
Successfully allocated 150000 to process P4
allocator>RQ P5 80000 B
No hole of sufficient size
allocator>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 = 128576]:
Address [150000:199999] len = 50000
Address [970000:1048575] len = 78576
allocator>C
Compaction process is successful
allocator>Status
Partitions [Allocated memory = 920000]:
Address [0:149999] Process P4
Address [150000:499999] Process P1
Address [500000:799999] Process P2
Address [800000:919999] Process P3

Holes [Free memory = 128576]:
Address [920000:1048575] len = 128576
allocator>
```

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

- `allocation.c`

GitHub Repository Creation Instructions:

To keep track of the assignment work, we will be using a GitHub Classroom-based repository. To submit a group assignment project on GitHub Classroom, follow these steps:

- Make sure that each member of the group has a GitHub account.
- Use the link https://classroom.github.com/a/j3c_jFG_ to access the assignment.
- Once you have accessed the assignment, you will see a green button that says, "Accept this assignment." Click on the button to create a repository for your group. Carefully select your name.
- Next, you will be prompted to choose a team for your group. If your team already exists, select it. Otherwise, click the "Create a new team" button to create a new team for your group. Include your teammate in the new team (only one of you will be creating the team). The team's name must be your first name and your teammate's first name, e.g., “A5-firstname1-firstname2”. Only one of you will create a team (named as mentioned), and the other will join it. Caution: Do not create a random name of your team.
- After creating the repository (by default, Private and keep it as is), each group member must clone it to their local machine using Git. To do this, navigate to the repository's page and click the green "Code" button. Copy the HTTPS link and use it to clone the repository. You can use GitHub Desktop on your local OS to manage the GitHub repository better.
- Work together to complete the assignment, ensuring everyone contributes and regularly pushing your work onto GitHub. Your contributions and division of the work will be graded based on the repo activities. If your GitHub account is missing from the commits/contributions list, you will get zero for the assignment.
- Keep pushing your changes daily to GitHub so I can check your progress. Once the assignment is complete, push the changes to the repository using Git.
- When you are ready to submit the assignment, navigate to the assignment page on Gradescope and click the "Submit assignment" button. This will prompt you to select the repository that you want to submit. Select the repository that your group created.
- Now, confirm that you want to submit the assignment. You will be asked to provide a comment explaining your submission. Provide relevant details and click on the "Submit assignment" button.
- In the final step, visit Gradescope and select the project assignment. It will ask you to connect your GitHub Account. Once authorized, you can select the repository you were working with and select the branch master to submit on Gradescope. Do not forget to add your assignment partner to the Gradescope Assignment.
- That's it! Your group assignment has now been submitted to GitHub Classroom. I and marker will be able to review your submission and provide feedback.