CP386: Assignment 4

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 be compiled 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.
- 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: 100):

In this project, you will develop an algorithm for contiguous memory allocation using one of the following strategies: first fit, best fit, or worst fit (refer to section 9.2 of the text). This project involves managing a contiguous region of memory of size MAX, where addresses range from 0 to MAX-1. Your program must respond to three types of requests:

1. **Request for a contiguous block of memory**
2. **Release of a contiguous block of memory**
3. **Report the regions of free and allocated memory**

Requirements:

1. **Memory Allocation**:
   - Implement memory allocation using any one of the first fit, best fit, or worst fit algorithms.
   - Keep track of different allocations and holes representing available memory.
   - Allocate memory from one of the available holes based on the selected allocation strategy. For the first allocation, all memory is available.
   - If there is insufficient memory to fulfill a request, output an error message and reject the request.

2. **Memory Management**:
   - Keep track of which region of memory has been allocated to which process. This information is necessary to support the 'Status' command and the 'RL' command (release memory).
   - When releasing memory, if a partition being released is adjacent to an existing hole, combine the two holes into a single hole.

3. **Program Invocation**:
   - Your program should be invoked by passing the initial size of the memory via the command line.

- Example invocation: ./allocation 1000000

4. **Program Execution Loop**:
   - The program will run a loop where the user enters commands. Entering "Exit" will stop its execution. The program will respond to the following commands for memory allocation, release, and status reporting:
     - **RQ <process number> <size> <B>**: 'RQ' command is for a new process that requires memory. It is followed by the amount of memory being requested and the algorithm flag. Use the following flags for different allocation approaches:
       - "F" for the first fit algorithm
       - "B" for the best fit algorithm
       - "W" for the worst fit algorithm
       - Note: You must mention the approach you have implemented.
     - **RL <process number/name>**: 'RL' command will release the memory that has been allocated to a process.
     - **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 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 affected by compaction.
     - **Status**: The 'Status' command is used for reporting the status of memory.
     - **Exit**: The 'Exit' command exits the loop and the program.

     - **Example Usage**:
       - A request for 20,000 bytes will appear as:

         ```
         command> RQ PO 20000 B
         ```

         Here, "PO" is the new process requesting 20,000 bytes using the best-fit algorithm.

       - A release will appear as:

         ```
         command> RL PO
         ```

5. **Implementation**:
   - Write all the functions required to implement the contiguous memory allocation algorithms.
   - Complete the program as per the described functionality.
   - Write all the code in a single C file.
   - To run the code, use the command make run, which initializes the program with 1 MB (1,048,576 bytes) of memory.

The expected output is given as below:

```
$ make runq2
gee -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 Pl 350000 B
Successfully allocated 350000 to process Pl
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 Pl
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 Pl
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 PS 80000 B
No hole of sufficient size
allocator>Status
Partitions [Allocated memory= 920000]:
Address [0:149999] Process P4
Address [200000:549999] Process Pl
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>(
Compaction process is successful
allocator>Status
Partitions [Allocated memory= 920000]:
Address [0:149999] Process P4
Address [150000:499999] Process Pl
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`