PROGRAMMING ASSIGNMENT 2

Subject : Stack, Queue, Dynamic Memory Allocation
Advisor : Asst. Prof. Dr. Burcu CAN, Asst. Prof. Dr. Adnan ÖZSOY, Assoc. Prof. Dr. Sevil ŞEN, Res. Assist. Burçak ASAL
Programming Language : C
Due Date : 13.11.2017

Introduction: By the help of this experiment, students will learn the basics of stack and queue structures and ultimately dynamic memory allocation concept.

Basic Client-Server Simulation

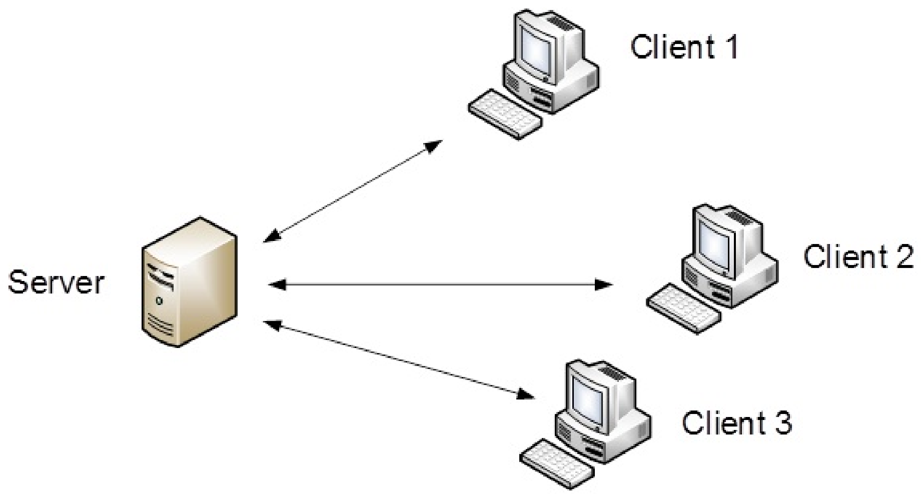


Figure 1: Schema of a basic client-server architecture

In this assignment, you will design a basic client-server architecture (Figure 1) in which, each client operates on a specific set of processes or takes requests (interrupts) from a user. The client sends the requests and processes to a server and similarly, the server takes the processes sent by the clients and its own interrupts .

Implementation Details

In more detail, each client should have a queue structure to operate a set of processes and have a stack structure for handling a set of interrupts from a user. Then the clients transmit their processes and interrupts to the server (More specifically, to the server's queue).

Similarly, the server should have also a queue structure to take and operate the processes of the clients and have a stack structure for handling its own interrupts.

Finally, the program should follow the details below, during the implementation stage;

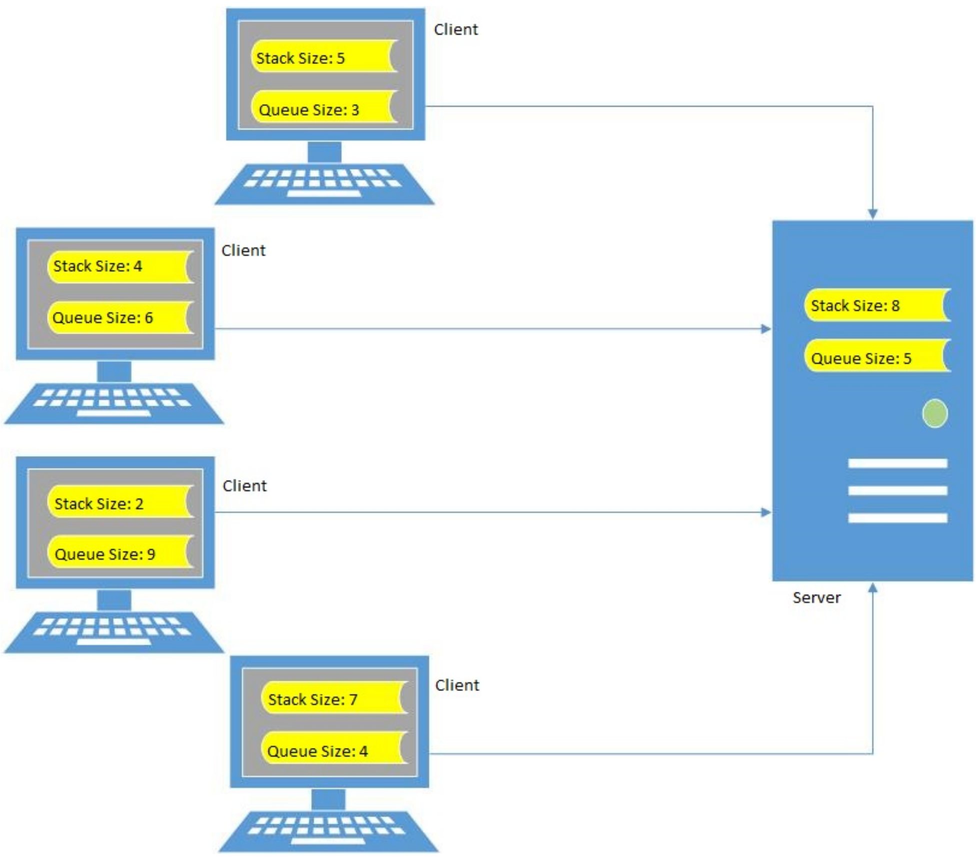


Figure 2: Sample schema for a basic client-server architecture with input parameters

1. Both for the clients and the server, the stack structure should always have more priority, compared to the queue structure. This means, before processing the client queue, the interrupts in the client stack should always be checked and sent to the server before the processes in the client queue. Similarly, the interrupts in the server stack should always be run before the processes in the server queue.

2. The latest interrupt in the client stack has to be sent first to the server and the latest interrupt in the server stack has to be run first.

3. In the architecture, there is always just one server but the number of clients varies in different input scenarios. Similarly, for both the clients (for each one) and the server, the stack size and queue size may also vary depending on the input situation. But again, the clients and the server will have just one queue and one stack structure, which is fixed. Thus, you should use dynamic memory allocation in your program for the variant factors above. For example, Figure 2 illustrates a specific input scenario in which there are four clients and a server and each client's and server's stack and queue size are (5,3), (4,6), (2,9), (7,4) and (8,5) respectively.

4. You should check and handle every possible different situation for stack and queue structures in every client and the server. If the client's stack is full, you will ignore the next interrupt command. If the client's stack is empty, you should continue to send the

current process in the client's queue. If the client's queue is full, you will ignore the next process command. Finally, if the client's queue is empty, you will give an error character (see item 8, for the error rules), when there is a send command. Same rules also apply for the server.

5. Clients can take numbers in range of (1, Client size), while the server takes a number in (Client size +1)
6. Processes and interrupts' names consist of lower case letters from English alphabet (between 'a' and 'z').
7. The program will handle four types of basic commands. Each command type consists of three slots :
   - **Process Request Command for the Clients:**

     Format: **'A' 'Client Number' 'Process Character Name'**

     In this command:
     - 'A' character in the first slot means that there will be an addition to a specific client's queue structure.
     - The second slot in this command specifies which client's queue will a process be added to. It takes a number in the range of (1, Client size).
     - The third slot in this command specifies the name of the process that will be added.
   - **Interrupt Request Command for Clients/Server:**

     Format: **'I' 'Client/Server Number' 'Interrupt Character Name'**

     In this command:
     - 'I' character in the first slot means that there will be an addition to a specific client's or the server's stack structure.
     - The second slot in this command type specifies which client's (or whether the server's) stack an interrupt will be added to. It takes a number in the range of (1, Client size+1).
     - The third slot in this command specifies the name of the interrupt that will be added.
   - **Send Command for the Clients:**

     Format: **'S' 'Client Number' 'G'**

     In this command:
     - 'S' character in the first slot means that from a specific client, a process or an interrupt will be sent to the server, more specifically, it will be added to the server's queue.

  - The second slot in this command type specifies from which client, a process or an interrupt will be sent to the server. It takes a number in the range of (1, Client size)
  - The third slot will always take 'G' character, which means an invalid slot and it can simply be ignored.

- **Operate Command for Server:**

  Format: **'O' 'G' 'G'**

  In this command:

  - 'O' character in the first slot means that the server will start to run its current process or interrupt.
  - The second and third slot will always take a 'G' character.

8. You should give three types of different error characters for three types of error cases:
   - If a specific client's queue is full, you should give an error character '1'. (During process addition)
   - If a specific client's stack is full, you should give an error character '2'. (During interrupt addition)
   - If a specific client's stack is empty, but also its queue is empty as well, then you should give an error character '3'. (During sending/operating a command for the clients/server)

   Same rules above also apply for the server's stack and queue structure. Please notice especially that, for the third rule, both the queue and the stack structures must be empty for giving the error character '3'. For a specific client or the server, when only the stack structure is empty, and the send/operate command is given for the clients/server, then you shouldn't give an error character for the stack structure. You should switch to queue and continue normally.

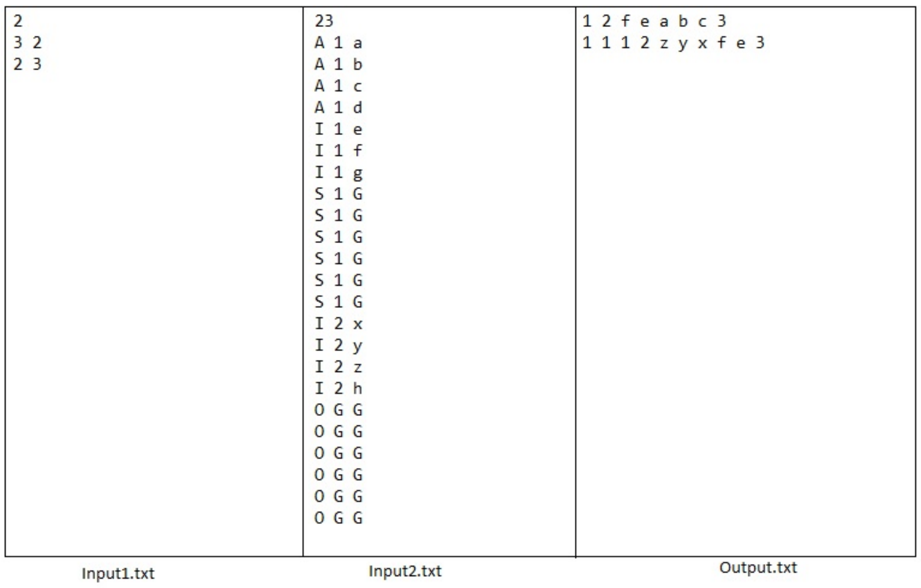
Input and Output

Your program should take three command line arguments (file paths) for two input files and an output file when it starts to run:

Run Format:

Program 'File Path for Input 1' 'File Path for Input 2' 'File Path for Output'

You will read the input from two input files and write your results to an output file (See Figure 3). As an output, you are expected to write a log history for each client and the server. The log history will have the history of each client/server's send/operate history and the error codes.



Input1.txt

```
2
3 2
2 3
```

Input2.txt

```
23
A 1 a
A 1 b
A 1 c
A 1 d
I 1 e
I 1 f
I 1 g
S 1 G
S 1 G
S 1 G
S 1 G
S 1 G
S 1 G
I 2 x
I 2 y
I 2 z
I 2 h
O G G
O G G
O G G
O G G
O G G
O G G
```

Output.txt

```
1 2 f e a b c 3
1 1 1 2 z y x f e 3
```

Figure 3: Sample Input and Output Files: **(Left):** First line in the Input1.txt specifies how many items (including the server, two in this case, one client and one server) the system consists of. In the second line, a specific client's queue and stack sizes are specified respectively. Last line (Third line, in this case) always specifies the server's queue and stack sizes, respectively. **(Middle):** First line in Input2.txt specifies the total number of the commands in the file. From the second line, the commands start. **(Right):** In Output.txt, by starting from the first line, you should write each client's log history (as one line consists of one client's history). Finally for the last line, you should also write the server's log history. Especially note that, for each line, the history starts from the left and finishes at right.

Report

1. Do not copy-paste from the experiment sheet.
2. Briefly explain what you understand from the problem.
3. Provide a detailed description of your solution.
4. Provide a detailed description of your functions and other code pieces.

Notes

- Do not miss the deadline.
- Save all your work until the assignment is graded.
- The assignment must be original, individual work. Duplicate or very similar assignments are both going to be considered as cheating.

- You can ask your questions via Piazza (`https://piazza.com/class/j7yfva1krme4gz?cid=15`) and you are supposed to be aware of everything discussed in Piazza.
- The submissions whose upload score is 0 will not be considered for evaluation.
- You will submit your work from `https://submit.cs.hacettepe.edu.tr/index.php` with the file hierarchy as below:
- The experiment code will be tested on the dev machine. Your source code should be compiled with GCC. Otherwise your experiment will not be evaluated.

This file hierarchy must be zipped before submitted (Not .rar, only .zip files are supported by the system)

```
→ <student id>
    → hw2.c (source code)
    → report.pdf
    → Makefile
```

This file hierarchy must be zipped before submitted (Not .rar , only .zip files are supported by the system).