Exercise 1: Message Queues

Two processes: client and server, communicate via two message queues "Up" and "Down".

1. The client reads a message from the standard input and sends it to the server via the Up queue, then waits for the server's answer on the Down queue. The server is specialized in converting characters from lower case to upper case and vice versa (N.B. the converter is done for you at the end of the document.) Therefore, if the client sends the message "lower case" via the Up message queue, the server will read the message, convert it, and send "LOWER CASE" via the Down queue. When the client receives the message from the server, it prints it out. You may assume the maximum size of any message is 256 bytes.

2. Multiple clients must be able to connect to the up and down queues. However, what happens if one client puts a message to convert on the system and another client is waiting for its response from the queue? There are different ways to handle this, but you should handle this using **"typed messages".** Each client has a particular client number based on the last 4 digits of its process ID. Use this 4- digit number as the client identifier and make it the message type so that each client can always be sure to receive the message that it is waiting on to be converted.

3. The server and client should be running forever using a while loop, however you can terminate the program using ctrl+c (SIGINT). In the server, when you terminate the program you should delete the message queues before termination using the SIGINT handler.

   Clarification:

   - When the user presses **Ctrl+C** in the client program, this means that you should terminate the client program **ONLY**. (this does not mean you destroy the message queue).
   - When the user presses **Ctrl+C** in the server program, this means that you should terminate the server program **ONLY**. (here, you need to destroy the message queue before termination).
   - When the server removes the message queue, the client will automatically fail to send anything on this queue (because it is already destroyed). This is completely normal.

   N.B.: You can run the server and multiple clients at the same time using different terminals. Grading: [4 marks for server, 4 marks for client, 2 marks for signal]

Exercise 2: Shared Memory & Semaphores

Two processes, client and server, communicate via message queues or semaphores, and shared memory segment.

1. The client reads a message from the standard input and writes it to the shared memory. Then it sends a message to the server asking it to process the data it wrote to the shared memory. The server is specialized in converting characters from lower case to upper case and vice versa (same as Exercise 1). Therefore, if the client wrote "lower case" in the shared memory, the server will read it, convert it, and write "LOWER CASE" to the shared memory. Then it sends a message to the client telling it that it finished processing the data. Write two C programs client.c and server.c that implement the behavior explained above. The server should serve one client at a time.

2. [Bonus] Repeat the previous experiment but the server should be able to handle up to N clients simultaneously trying to read and write in the same shared memory segment.

N.B.: You can run the server and multiple clients at the same time using different terminals.
Grading: [5 marks for server, 5 marks for client]

Deliverables:

1. **Source codes** as a zipped folder as following:
    a. Section#_Bench#.zip (for semester) and Code#.zip (for credit) containing two directories as follows:
        i. Exercise1 (contains source code for Exercise 1)
            1. client.c
            2. server.c
        ii. Exercise2 (contains source code for Exercise 2)
            1. client.c
            2. server.c
        iii. **All other files that you used for each exercise** that will allow us to run your code on our side. (Place these files in each directory).

Notes:

1. Write neat code with comments (whenever possible).
2. We will not perform automated testing on this assignment, so you do not need to care about I/O format. However, it's your responsibility to make sure that you are printing the correct messages.

You may want to use this conversion function for the server code:

Code for conversion from uppercase to lowercase or vice versa:

```
#include <ctype.h>
/* convert upper case to lower case or vise versa */
void conv(char *msg, int size) {
    int i;
    for (i=0; i<size; ++i)
        if (islower(msg[i]))
            msg[i] = toupper(msg[i]);
        else if (isupper(msg[i]))
            msg[i] = tolower(msg[i]);
}
```