Inter-process Communication (IPC)

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 reversing strings. (N.B. The reversing code is done for you at the end of the document). Therefore, if the client sends the message "Hello World" via the Up message queue, the server will read the message, reverse it, and send "dlroW olleH" 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 reverse 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.

Exercise 2: Shared Memory & Semaphores

Replicate the previous exercise but using **shared memory** and **semaphores** ONLY. You should not use **ANY message queues** in this exercise.

1. The client reads a message from the standard input and writes it to the shared memory where it will be reversed by the server. When the server is done with reversing the string, the client should print the reversed string. Write two C programs client.c and server.c that implement that behavior. 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. (N can be any arbitrary number of clients).

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 free the shared memory and destroy any semaphores before termination using the SIGINT handler.

Deliverables:

Source codes as a zipped folder as following:

1. Section#_Bench#.zip (for semester) and Code#_Name.zip (for credit) containing two directories as follows:
    a. Exercise1 (contains source code for Exercise 1)
        i. client.c
        ii. server.c
    b. Exercise2 (contains source code for Exercise 2)
        i. client.c
        ii. server.c

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.
3. You can run the server and multiple clients at the same time using different terminals.

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

Code for reversing a string:

```
#include <string.h>
/* reverse a message */
void reverse(char* msg, char* reversed) {
    int i;
    for (i=0; i<strlen(msg); ++i)
        reversed[i] = msg[strlen(msg) - i - 1];
}
```