EECS 338 Homework 4: Interprocess Communication

General requirements:

- See posted due date.
- Create a typed report for discussion questions and program output.
- Upload a single, compressed file (e.g. zip) to Canvas that contains all required files (programs + report).
- All work should be your own, as explained in the Academic Integrity policy from the syllabus.
- Include either a single makefile that compiles all programs or a separate makefile for each.

1. Write two programs that play a trivia game using shared memory. **Include examples of your program output in a report.** In order to synchronize the question-answer-result sequence, you will force processes to sleep. Create separate server and client programs (two .c files) according to the following design requirements:

   - The server is assumed to be executed *before* the client.
   - The server creates the shared memory and writes a string to shared memory containing the trivia question and sleeps for some period of time (e.g. 10 seconds) to allow the client to get user input.
   - The client, which is started *after* the server, reads and displays the question from shared memory.
   - The client allows the user to type an answer and writes it to the *same* shared memory location, overwriting the question, and sleeps for some period of time (e.g. 10 seconds) to wait for the result from the server.
   - After the server's sleep period, it reads the client's answer, displays it, and compares it to the correct answer (which is known by the server). The server writes the result message (correct or incorrect) to the *same* shared memory location.
   - After the client's sleep period, it reads the result and displays it to the user.

   Below are examples of how your output might look (user input is in **bold**). The trivia question in both examples is "What is the capital of Greece?". In example #1, the client's user types the correct answer "Athens". In example #2, the client's user types the incorrect answer "London".

Server example #1:	*Client example #1:*
`Sent question. Sleeping...`	`What is the capitol of Greece?`
	`Athens`
`Answer received: Athens`	`Waiting for result...`
	`Correct!`
Server example #2:	*Client example #2:*
`Sent question. Sleeping...`	`What is the capitol of Greece?`
	`London`
`Answer received: London`	`Waiting for result...`
	`Wrong`

2. Write two programs that play a trivia game using a socket. **Include examples of your program output in a report.** If you wish, you may use a fixed port number and IP address of 127.0.0.1. Create separate server and client programs (two .c files) according to the following design requirements:

   - The server is assumed to be executed *before* the client.
   - The server creates a socket. When the client connects, the server writes the trivia question and attempts to receive the answer. Note that "accept" and "read" are blocking functions, as explained in class.

- The client, which is started *after* the server, reads and displays the question.
- The client allows the user to type an answer and writes it to the socket.
- The server reads the client's answer, displays it, and compares it to the correct answer (which is known by the server). The server writes the result message (correct or incorrect) to the socket.
- The client reads the result and displays it to the user.

Below are examples of how your output might look (user input is in **bold**). The trivia question in both examples is "What is the capital of Greece?". In example #1, the client's user types the correct answer "Athens". In example #2, the client's user types the incorrect answer "London".

Server example #1:	*Client example #1:*
`Sent question.`	`What is the capitol of Greece?`
	`Athens`
`Answer received: Athens`	`Waiting for result...`
	`Correct!`
Server example #2:	*Client example #2:*
`Sent question.`	`What is the capitol of Greece?`
	`London`
`Answer received: London`	`Waiting for result...`
	`Wrong`

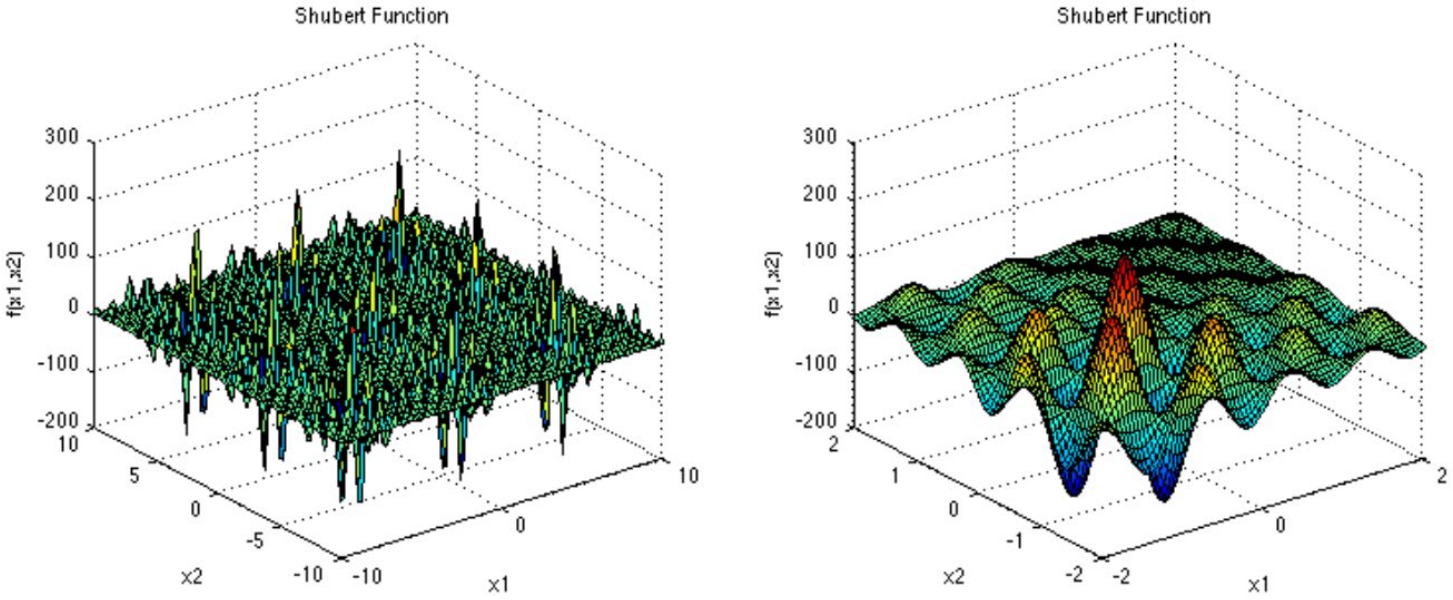
3. The program "shubert.c" demonstrates the use of the Shubert function, which is a test case for optimization algorithms (see Appendix). Using two processes and the communication methods below, print the minimum values (local and global) of the Shubert function for $-2.0 \leq x1,x2 \leq 2.0$ and the run time of the parent/server. Use a step size for both loops to yield a run time of *at least 1 second.* Compare your run time to that of the sample program using the same step size. Each process should perform an equal amount of work to find a local minimum in its range. For example, one process can do half of the x1 values (-2.0 to 0.0), and the other process can to the other half. You don't need to modify the other loop. The parent process can determine which of the two local minima is the least and print it as the global minimum. **Include the output in your report for each of the three programs and the sample program (shubert.c) using appropriate step size.** Below is an example of output for one program:

```
Total time was 10.65 seconds.
Local min = -62.12412
Local min = -186.73064
Global min = -186.73064
```

   a. Create a single program that uses fork() and shared memory. Tip: you can create shared memory with a value of type double and access the value using the dereference operator *. The program "shared_double.c" is provided as an example.
   b. Create a single program that uses fork() and a pipe.
   c. Create a either a single program or separate programs that use(s) a socket. Tip: there are two ways you can work with the double value. One way is to use an array of doubles with only one value. Another way is to declare a regular double and use the addressof operator & for read() and write().

Appendix: The Shubert function

The Shubert function can be used as a test case for optimization algorithms. It is a function of two variables, x1 and x2, which can represent 2-D coordinates in a plane. Plotting the function output as the height in 3-D shows many local minima and maxima. See the figure below.



$$f(\mathbf{x}) = \left(\sum_{i=1}^{5} i\cos((i+1)x_1 + i)\right)\left(\sum_{i=1}^{5} i\cos((i+1)x_2 + i)\right)$$

The Shubert function definition and plots using different parameter ranges. Left: -10 to 10. Right: -2 to 2. Figure is taken from http://www.sfu.ca/~ssurjano/shubert.html.

The goal of an optimization algorithm would be to find a *global* minimum or maximum, and normally we would want to use an efficient algorithm. The sample program "shubert.c" provides a brute force search using the range -2 to +2 with a step size of 0.5. Note that shubert.c must be compiled with the math library (-lm), as shown in the accompanying makefile. A step size of 0.5 is too large to provide a reasonable search, but it allows us to print all of the values in the range $-2.0 \le x1,x2 \le 2.0$. If you run the program, you should see that a minimum of -97.27 occurs at (x1, x2) = -1.5, -1.0. Note that you should not print the y values for your assignment using a small step size (there will be too many!).

In a brute force search, increasing accuracy often means having a longer run time. Using concurrency, we can divide the work across two processes by doing half of the search in each process. On a multicore computer, we expect this to be faster, perhaps even close to twice as fast. You might consider how much you can increase the accuracy using this approach with the same run time.

Interested students can see more information graphs of the Shubert function here:
http://profesores.elo.utfsm.cl/~tarredondo/info/soft-comp/functions/node28.html