EECS 338 Homework 6

General requirements:

- Due on the posted due date.
- Upload a single, compressed file (e.g. zip) to Canvas that contains all required files.
- Include either a single makefile that compile all programs or a separate makefile for each.
- **Include a typed document with the program output for all problems.**
- All work should be your own, as explained in the Academic Integrity policy from the syllabus. File sharing is prohibited.

Instructions: The following problems require you write programs that find the minimum of the Shubert function (see previous HW). You will use three different approaches (OpenMP, POSIX threads, and fork()). The following are additional requirements:

(a) Use two threads/processes (one parent + one child) such that each searches half of the overall x_1 range.
(b) Both threads/processes must store the minimum value in the same variable, and updates to the variable must be protected as a critical section. Note that all other sections of the program should not be treated as part of the critical section.
(c) Each thread/process should print the local x_1 range that it evaluates.
(d) The parent/master should print the final, global minimum.

Below is an example of possible output:

```
Parent x1 = -2.0 to 0.0
Child x1 = 0.5 to 2.0
minimum = -97.27
```

Include a typed document with the program output for all problems. Use the following approaches:

1. Use OpenMP and the "critical" clause for the critical section. You may need to use the Linux command "export OMP_NUM_THREADS=2" to control the number of threads. (If desired, you may use more than 2 threads.) You can use the thread ID to determine the x_1 range. See the tip below if you wish to try using "parallel for". Mac users should use a Linux server (see tip below).

2. Use POSIX threads and a semaphore for the critical section. Use a single, global variable to store the minimum.

3. Use "fork" with shared memory and a semaphore for the critical section. Use a single, shared variable to store the minimum.

Tip (known problem with Macs): There are known issues with OpenMP on Mac OS. We currently recommend just using our servers (eecslab-1.case.edu, etc.) for problem #1. Some people have had success with different solutions, including LLVM (https://clang-omp.github.io/) and building gcc from source. However, we cannot provide much, if any, support for these.

Tip (can I use OpenMP's "for"?): Yes, you can use "parallel for" to automatically assign x1 values. However, "parallel for" only works with integer iterator variables, so you would need to compute doubles for the x1 values using integer values (0*step, 1*step, 2*step, etc.).

Tip (semaphore and threads): The OpenMP program does not require a semaphore. When using POSIX threads, the semaphore should be a global variable.

Tip (memory mapping): When using “fork” with shared memory, you will need to share two variables: the semaphore and the global minimum. This requires a separate, shared memory space for every shared variable, using a different, unique name for each memory space, such as “Semaphore” and “Minimum”. If you dislike duplicating the code for each shared variable, you can create a helper function to do it.

Tip (using “&”): You aren’t required to understand the address-of operator “&”. However, pay close attention to the sample programs from class. The “&” is not required when the semaphore variable is a pointer. The simple rule is that “fork” and shared memory do not require “&” for the semaphore, but the global variable in the pthread example program does require it.

Rubric is forthcoming...