COSC 2P95 – Lab Exercise 9 – Signals, Mutexes, and Threads

Lab Exercises are uploaded via the Sakai page.
Since you need to submit both sample executions and your source files, package it all up into a .zip to submit.

For your lab task, you wrote a multithreaded application that could do some number-crunching.
Specifically, you computed hash functions.

However, we can do something a little more interesting than that.

Function Optimization

Several mathematical functions take multiple parameters and, for some defined domain, can have widely varying generated values.
To *optimize* a function is to find a set of parameters that aims to approximate the global *minimum* or *maximum* (depending on what you're searching for) of that function.

For example, if you wish to *minimize* a simple function like $y=x^2$, for a domain of $-5 \le x \le 5$, there's a clear global minimum of x=0.

However, several more complicated functions have numerous *local minima*, and those minima (combined with a more complicated function) can make it trickier to easily identify the true global minimum.

To some extent, *discretization* (remember our third lab exercise?) can help us here. However, typical discretization will scan the entire domain with *uniform spacing*. Conversely, if we find a potentially good area within the domain, we'd actually want to further study that area with a far higher precision (possibly ignoring the rest of the domain entirely).

Naturally, all of this also applies to *maximization*; it's just the direction that changes.

The Egg Holder function is an interesting one. The generalized formula is scalable for multiple dimensions, and is written as:

$$f(x)=\sum_{i=1}^{m-1}\left[-(x_{i+1}+47)\sin\sqrt{|x_{i+1}+x_i/2+47|}-x_i\sin\sqrt{|x_i-(x_{i+1}+47)|}\right]$$

However, since we really just want two axes (x and y), we can more simply write it as:

$$f(x,y)=-(y+47)\sin\left(\sqrt{|x/2+y+47|}\right)-x\sin\left(\sqrt{|x-y-47|}\right)$$

The domain we'll use is $-512 \le x,y \le +512$.

Hill-Climbing

Let's consider a hypothetical *local search*. A local search is a basic mathematical search that, when presented with different possible moves in some search space, picks the move that appears to yield an improved outcome.

For example, if we were minimizing $y=x^2$, and our current guess was x=0.2, then two potential new guesses might be 0.15 or 0.23. Since 0.15 would yield a preferable y value, that is what we would choose. We call it hill-climbing because, if we were trying to maximize, it would always choose the short-term decision that offered the highest immediate height.

Hill-climbers are *very* susceptible to local minima, so you typically need some additional tricks for them to be useful.

Parallel Hill-Climbers

Since a single hill climber isn't very intelligent, we can achieve better results by allowing multiple hill climbers to search in parallel. We'll achieve this by assigning a thread to each climber.

Thus, we shall formalize our hill-climbing algorithm as follows:

- We shall have a global concept of “best answer so far”
  - This will include the lowest minimum found so far, as well as the x and y that yielded that value
  - You may want to initialize this lowest minimum to something artificially high, and leave it to the climbers to immediately initialize it properly for you
- The user will be presented with a menu, asking for the number of climbers (threads) to run simultaneously
  - You can assume a maximum of 8 climbers/threads
  - If the user selects 0, the program exits
- Each climber has a current position (with corresponding current calculated height)
- Each climber will generate, at each step, 4 possible moves
  - The best possible move (i.e. the one that generates the lowest calculated value) is the possible next move for that climber
  - If the generated value would improve on the climber's current calculated height, make the move
    - Otherwise, randomize the position of the climber, within the established [-512..+512] domain, and recalculate its height
- Whenever any climber finds a new global best minimum, it updates the global “best answer so far”
  - Of course, this will require some form of *mutual exclusion*
- When the user presses ctrl+c, execution suspends, and the user is presented with the menu again
  - If the user chooses to pick a number of threads to try again, the “global best” thus far is not forgotten
- At the very least, whenever the user presses ctrl+c to suspend, you must show the “global best” height (and corresponding x and y) thus far. You may wish to also include support for SIGUSR1, to display the current progress without suspending the search

Tips:

- You can use `std::abs`, `std::sqrt`, and `std::sin` if you include the `cmath` header
- How you generate your possible moves from a given position will heavily influence the ability of the climber to improve upon its results
  - Consider generating two random modifiers (one for x and one for y), each in the range of [-5.0..5.0], and adding them to the coordinates. Then, if they've exceeded the bounds (-512..512), then clamp
- In practice, you're very likely to solve this function incredibly quickly. It isn't terribly difficult (particularly since we're using the variation with only two parameters)
  - If you're interested in seeing how much the threading and hill climber actually helps, feel free to widen to the generalized form of the function, which allows for more dimensions
- Make sure you're careful to use multiple mutexes. You certainly don't want two components printing simultaneously, and you *especially* don't want two threads updating the global idea of coordinates and best calculated value simultaneously!
- You may assume *sane input*:
  - The user will only enter numbers
  - The number will be a zero to quit, or a number from one to eight
  - You don't need to handle any signals other than SIGINT (and SIGUSR1, if desired)

Requirements for Submission:

For your submission, bundle a sample execution or two along with your source files. Pack it up into a .zip to submit (through Sakai).