Concurrency and Parallelism

GEI 2024

Lab 1 – Counters

We have a program in which several threads share two counters: `increment`, which starts at 0, and `decrement`, which starts at a given `total`. Each thread will loop `iterations` times, substracting 1 from `decrement`, and adding 1 to `increment`. At every step the sum of the two variables should be equal to `total`.

The program accepts the following options

- `-i n`, sets the number of iterations.
- `-t n`, sets the number of threads.
- `-a n`, sets the size of the array for part 3.

We provide you with the code that parses the options, and starts the threads. If you test the program you will see that it does not correctly protect the variables, and therefore the sum at the end is not equal to total. Add the following features:

Part 1 (**Protect the counters**)
Protect the counters using a mutex. Do not use global variables.

Part 2 (**Iterations**)
Change the behaviour of the iterations so that it specifies the total number of iterations performed between all the threads. That is, if the number of iterations is set to 100, the total number of increments and decrements should be 100.

If the program is working correctly, the `increment` variable should be equal to iterations at the end.

Hint: Count the iterations using a shared counter.

Part 3 (**Change the counters to arrays**) Change the counters from a single value to an array. The decrement counters should all be initialized to `total`. Each thread now randomly selects an increment position and a decrement position.

To increase concurrency, lock each individual counter with a different mutex, that is, increment[0] should use a different mutex than increment[1]. If done correctly, the sum of the values of the increment array plus the sum of the values of the decrement arrays should be `total * array size` at any given time. Make sure that at no point a different result can be observed, even when the threads are running.

Part 4 (**Increment shift**) Add a new type of thread that chooses two random positions in the increment array, decrements the first one, and increases the second one. This should leave the sum of the increment array unchanged. As in the previous exercise, make sure that, when observed by other threads, the sum of the array of increments plus the sum of the array of decrements is always `total`.

The number of threads in the option structure sets the number of threads of each type. That is, if the number of threads in the options is 10, we should have 10 threads for the behaviour of parts 1-3, and another 10 for this exercise. All of them should be running simultaneously.

The argument structures for these threads should be stored in the stack of the initial thread.

Part 5 (**Decrement shift**) Do as in the previous exercise, but for the decrement array. In this exercise, the argument structures for these threads should be stored in the heap.

Submission

The submission deadline is February 18. The initial code is available at github classrom at https://classroom.github.com/a/W8HddegP. When you first access the assignment you will be able to create your lab group. Name your group using the logins of the members separated by a hyphen. Once the first member of the group has created the repository, you should be able to add the second member.