CS 475/575 -- Spring Quarter 2023

Project #3

The Mutex Stack Challenge

100 Points

Due: May 10

This page was last updated: April 19, 2023

Introduction

A *stack* is a Last-In-First-Out data structure. If you have forgotten about stacks since CS 261, there are many good reviews on the Internet. Here is one of them.

In our full **OpenMP** noteset, we looked at how to use OpenMP Mutexes to keep two threads from colliding when trying to do operations using the same variable. A great example of this is implementing a stack.

```
int             Stack[NUMN];
volatile int    StackPtr = -1;
                // to push: increment stack pointer, then place number here
                // to pop:  take number from here, then decrement stack pointer
```

Project Requirements:

1. Start with this sample code, mutex03.cpp. It is already setup to fire off the PushAll and PopAll threads using OpenMP parallel sections. It does not (yet) implement mutexes.
2. The Makefile I used looks like this:

```
mutex03:                mutex03.cpp
                        g++ mutex03.cpp  -o mutex03  -lm -fopenmp
```

3. Run the program and note its behavior.
4. The **omp_lock_t** variable called *Lock* has been declared, but not initialized. Initialize it in the right place.
5. Change the defined constant USE_MUTEX to true.
6. Add the proper Mutex code inside if( ) statements like this:

```
if( USE_MUTEX )
{
        . . .
}
```

   Do this in as many places as are necessary. In the interest of performance, lock the mutex as late as you can and unlock it as early as you can. (That is, don't mutex the entire source code...)
7. Run the program again. Note the program's behavior now.
8. ***This is not a performance-graphing project.*** You will use just the two threads. There are no performance tables or graphs to do.
9. Vary the size of the stack, i.e., the **NUMN** defined constant.
10. Also vary how many times you try it with mutexes turned on and mutexes turned off. I ended up using a script that looked like this:

```
#!/bin/bash
for n in 1024 2048 4096 8192 16384 32768
do
  for m in true true true true true false false false false false false false false false false
  do
       g++  mutex03.cpp  -DNUMN=$n -DUSE_MUTEX=$m  -o mutex03  -lm  -fopenmp
      ./mutex03
  done
done
```

   This is what I did, but, you can handle the requirements any way you please.

Commentary:

Your commentary write-up (turned in as a PDF file) should include:

1. Tell what machine you ran this on
2. Tell what operating system you were using
3. Tell what compiler you used
4. Include, in your writeup, the pieces of code where you implemented the mutexes
5. Tell us what you discovered by doing this:
    1. Does the non-mutex way of doing this *ever* work? If so, how often?
    2. Does changing NUMN make any difference in the failure percentage?

3. Is there a difference in elapsed execution time between mutex and non-mutex? Why do you suppose this is? (Ignore the very large elapsed times -- these are a reult of the TIMEOUT being used up.)

6. No tables or graphs are needed for this project.

A Summary of OpenMP Mutex Declarations and Functions

- omp_lock_t Lock;
- omp_init_lock(&Lock);
- omp_set_lock(&Lock);
- omp_unset_lock(&Lock);

Grading:

Feature	Points
Correctly implemented the mutexes in the correct places	30
Ran the code for a variety of NUMN values with mutexes in use	20
Ran the code for a variety of NUMN values with mutexes not in use	20
Commentary (see topics above)	30
Potential Total	**100**