Prof. Jingke Li (FAB 120-06, lij@pdx.edu); Class: TR 16:40-17:55 @ ASRC 230; Lab: F 10:30-11:50 @ FAB 88-10.

Lab 2: Programming with Pthreads

Download and unzip the file `lab2.zip` from D2L. You'll see a `lab2` directory with some program files.

1. Condition Variables

The program file `condvar-pthd.c` is an incomplete Pthreads program. The `main()` routine creates two threads, one sender and one receiver. Complete the program by providing code for these two threads, so that the sender will send a signal, and the receiver will wait for the signal.

The `sleep(1)` call in `sender()` is to help to see the waiting. When you run the completed program, you should see two message lines first:

```
Sender starts ...
Receiver starts ...
```

After a pause, you should see the third line:

```
Signal received!
```

2. Array Sum

The program file `arraysum-pthd.c` is a simple Pthreads program for computing array sum. (We've shown this program in class.) Use an editor to open the file; read and understand the program; and then compile and run it:

```
linux> make arraysum-pthd
linux> ./arraysum-pthd
...
The sum of 1 to 1000 is 500500
```

Exercise 1 In class, we showed additional versions for this program. Your task is to create and run these versions. Specifically,

1. Modify the `printf` statement to also print out the CPU id, along with the thread id.
2. Insert the CPU-affinity code shown in class to the program, so that thread k is assigned to run on CPU k. (If there are more threads than CPUs, then follow a round robin assignment.)
3. Add command-line arguments for `arraySize` and `numThreads` configurations.

Note that you need to change the inclusion headers accordingly for each step. You may want to stop and test the program after each step.

Exercise 2 Assume you have a working version of the modified program. Comment out all the locking and unlocking statements in the program; and re-compile it. Now, this program has the potential for race conditions. Run the program with different `arraySize` and `numThreads` configurations until you see an evidence of a race condition occurring. Write down the array size, the number of threads, and the evidence.

3. New Pthreads Programs

The program file `mtxmul.c` contains a simple sequential implementation of matrix multiplication. Use an editor to open the file; read and understand the program; and then compile and run it.

1. Convert `mtxmul.c` into a Pthreads program, `mtxmul-pthd.c`. Use `N` threads, one for each iteration of the `i` loop. Compile and test your program.

2. Write a second version of the Pthreads program, `mtxmul2-pthd.c`. In this version, the number of threads is not fixed. The program reads in an optional command-line argument representing the number of threads. If the argument is not provided, the program use the default value of `N`. For example,

   ```
   linux> ./mtxmul2-pthd 4    // using 4 threads
   linux> ./mtxmul2-pthd      // using the default of N threads
   ```

   Use a simple partitioning scheme to partition the workload for the threads.