

4.1. Modify the mutex version of the  $\pi$  calculation program so that the critical section is in the for loop. Compare the performance of your version with the serial version.

4.2. Modify the mutex version of the  $\pi$  calculation program so that it uses a semaphore instead of a mutex. Compare the performance of your version with the serial version.

4.3.

Although producer-consumer synchronization is easy to implement with semaphores, it's also possible to implement it with mutexes. The basic idea is to have the producer and the consumer share a mutex. A flag variable that's initialized to `false` by the main thread indicates whether there's anything to consume. With two threads we'd execute something like this:

```
while (1) {
    pthread_mutex_lock(&mutex);
    if (my_rank == consumer) {
        if (message_available) {
            print message;
            pthread_mutex_unlock(&mutex);
            break;
        }
    } else { /* my_rank == producer */
        create message;
        message_available = 1;
        pthread_mutex_unlock(&mutex);
        break;
    }
    pthread_mutex_unlock(&mutex);
}
```

- a. Write a Pthreads program that implements this version of producer-consumer synchronization with two threads. Suppose there is only at most one message at all time.
- b. Generalize this so that it works with  $2k$  threads – odd-ranked threads are consumers and even-ranked threads are producers. Every odd-ranked thread must indicate that it receives message and also indicate which thread sends the message.

```
Th 5 > message: hello from 0
Th 3 > message: hello from 4
Th 1 > message: hello from 2
```

- c. Generalize this so that each thread is both a producer and a consumer. In particular, suppose that thread  $q$  “sends” a message to thread  $(q + 1) \bmod t$  and “receives” a message from thread  $(q - 1 + t) \bmod t$ , where  $t$  is the total number of threads. Every thread must indicate that it receives message and also indicate

which thread sends the message.

```
Th 1 > Received: hello from rank 0  
Th 2 > Received: hello from rank 1  
Th 3 > Received: hello from rank 2  
Th 4 > Received: hello from rank 3  
Th 0 > Received: hello from rank 4
```

Templates for 4.1 and 4.2 are provided, they both use timer.h to obtain the system time, and this head file is also provided.

**Note: All programs have to be presented to the instructor on the class of due date.**