COEN 11 - Homework - Threads

Answers

1. Give an example of a situation in which using different threads for different tasks would be helpful.

```
>> text editors, web servers, etc.
```

2. **Splitting the work** -- calculate the value of Π using N threads:

$$\Pi = 4\ (1 - 1/3 + 1/5 - 1/7 + \ldots + (-1)^n\ 1/(2n+1))$$

```
Algorithm:
double factor=1.0;
double sum=0.0;
for (i = 0; i < n; i++, factor = -factor)
      sum += factor / (2*i + 1);
pi=4*sum;
```

```
// not very effective use of locks
// sum is global
void* thread_sum(void *id)
{
     int myid = (int)id;
     int i;
     int mysize  = n / thread_count;
     int mystart= mysize * myid;
     int myend = mystart + mysize;
     double factor;

     if (mystart % 2 == 0)
          factor = 1.0;
     else
          factor = -1.0;

     for (i = mystart; i < myend; i++, factor = -factor)
     {
          pthread_mutex_lock (&mutex);
          sum += factor / (2*i + 1);
          pthread_mutex_inlock (&mutex);
     }

     // pi = 4 * sum; -- should happen after all threads are done
     return NULL;
}
```

```
// more efficient use of locks
// sum is global, but my_sum is local
void* thread_sum(void *id)
{
     int myid = (int)id;
     int i;
     int mysize  = n / thread_count;
     int mystart= mysize * myid;
     int myend = mystart + mysize;
     double factor;
     double mysum = 0.0;

     if (mystart % 2 == 0)
          factor = 1.0;
     else
          factor = -1.0;

     for (i = mystart; i < myend; i++, factor = -factor)
             my_sum += factor / (2*i + 1);

     pthread_mutex_lock (&mutex);
     sum += my_sum;
     pthread_mutex_inlock (&mutex);

     // pi = 4 * sum; -- should happen after all threads are done
     return NULL;
}
```

3. **Splitting the data** -- write a thread function to initialize int array x so that each element receives its index in the array: x[i] = i, and each thread initializes its portion of the array. Note that i relates to the entire array. The size of the array is N, and your program will execute with nthreads (which is a global value). Each thread receives an id between 0 and nthreads-1. Assume N is a multiple of nthreads.

```
void
init (void *arg)
{
  int id = (int)arg;
  int mysize = N / nthreads;
  int mystart = id * mysize;
  int myend = mystart + mysize;

  for (i = mystart; i < myend; i++)
       x[i] = i;

  return NULL;

}
```

4. **Splitting the data** -- write a thread function to initialize int 2D array x (NxN) so that each thread initializes its portion with i+j in each slot. Note that i and j relate to the entire array. Each thread operates on a strip independently, and your program will execute with nthreads (which is a global value). Each thread receives an id between 0 and nthreads-1. Assume N is a multiple of nthreads.

```
void
init (void *arg)
{
  int id = (int)arg;
  int mysize = N / nthreads;
  int mystart = id * mysize;
  int myend = mystart + mysize;

  for (for (i = mystart; i < myend; i++)
       for (j = 0; j < N; j++)
            x[i][j] = i + j;

  return NULL;

}
```