Problem for 31 July 2019

In /u/csc209h/summer/pub/ex/11/starter.c you will find code for a basic merge sort. This program also initializes an array with random numbers before invoking the sort, and does some checking after the sort to see if it worked.

You don’t have to read all of the code, but do note the mysort() function and how it is recursive—see the initial call in main(), and see the two recursive calls in mysort() itself. (Ignore mysort()’s bizarre return type for now; this will save you time later.)

Today’s problem is to make this into a parallel merge sort, by using threads.

One issue is that thread functions only take one argument, but we have two arguments to mysort(). So to make a threads version of this code, we need to change it to use a pointer to a struct. Add this struct type declaration at the top:

```c
struct args {
    int *p;
    int size;
};
```

and change mysort() to take a single argument of type “void *” (it’s already defined as returning type “void *” as is required for posix threads).

Then change the three mysort() calls (one in main(), two in mysort()) to create a “struct args” and pass a pointer to that instead. Example:

```c
struct args subarg1 = { p, midpoint };
mysort(&subarg1);
```

mysort will then need to unpack its arguments like so, at the beginning of mysort():

```c
int *p = ((struct args *)arg)->p;
int size = ((struct args *)arg)->size;
```

Get this working without threads before proceeding. (Remembering that “working” includes compiling with “gcc –Wall” with no warnings. And don’t use any casts other than the above two.)

Then, make it create threads. Instead of recursively calling one mysort() and then the other, make it do the two mysort()s in parallel using threads. But you can’t proceed to the merge until both of the sub-sorts have completed.

And, of course, test that it’s still working.

If you have further time, you can read and ask about more of the code.

Reminders of pthread library function call parameters:

```c
extern int pthread_create(pthread_t *t, pthread_attr_t *attr, void *(*func)(void *), void *arg);
```

example call:

```c
pthread_create(&t, NULL, myfunction, &data)
```

```c
extern int pthread_join(pthread_t t, void **returnvalue);
```

example call:

```c
pthread_join(&t, NULL)
```

(Unfortunately, in its current form this code is not amenable to timing tests, because acquiring the random numbers takes a significant amount of time in comparison to the sort, and, more importantly, takes a highly variable amount of time depending on the randomness pool in the OS kernel which is used for /dev/random, and furthermore running this program a lot will deplete the randomness pool so it will affect the timing even more. All this is to say that to do timing tests you’d want to split the randomization from the sort, and repeat only the sort many times.)