Homework 4 Write-Up

Aaron Okano, Jason Wong, Meenal Tambe, and Gowtham Vijayaragavan

May 18, 2011

Write a COGENT explanation, along the lines of the "Gowtham and Noelle" story in class, of how the threads take turns in this application. Explain why it may be useful to have more threads than CPU cores for this program but not for the prime finder above.

In the program, we needed to create a web probe that would read through a list of URLs in a file using the number of threads that the user specifies. The file—in our case, urls.txt—contains a list of URLs to probe. The program takes three arguments, to be given in the following format:

```
% webprobe <URL file> <window width> <number of threads>
```

The output of the program consists of the mean and standard deviation of the **window width** most recent probe times and the number of accesses performed by each thread.

The **main()** function begins by initializing all the variables and arrays in accordance with the arguments. Then the "probe" threads, which each work to probe a URL, are created as well as the "reporter" thread. Afterwards, it simply waits for the threads to finish—a condition which is not met, since the threads immediately begin an infinite loop.

When each "probe" is created, it begins execution of the **probe()** function. Because several threads will be accessing and writing to the same sets of data (namely, the file pointer and the list of most recent accesses) **probe()** makes use of mutual exclusion objects (mutexes) which to coordinate the threads. The mutexes themselves act as locks, which, after locked by the first thread to encounter **pthread_mutex_lock()**, prevent other threads

from executing the code following that point until the mutex is unlocked. The actual probing and recording of access times is performed by first recording the time of day, forking a `wget` process, recording the time of day after `wget` exits, and finally, finding the difference between the two times and inserting the value into an array of the **window width** most recent times using the function **load_recent()**. The function then continues to loop this cycle until termination.

The single "reporter" thread, as can probably be guessed, executes the **reporter()** function. This function is set to sleep for 10.0 / number of threads seconds using the usleep command. After sleeping, it first locks the mutex to prevent the probe threads from accessing the data it needs to read. Then it prints the mean and standard deviation of the most recent accesses along with the number of accesses made by each probe thread. As with the probes, **reporter()** continues to loop until the program's termination.

Compared with the Primes program, webprobe benefits from having more threads than available cores. The reason for this is that while each thread of the Primes program runs computations the entire time, much of the time a probe is executing, it is only waiting for `wget` to complete execution. Rather than waste all that time waiting, the CPU can be put to use launching and timing more instances of `wget`.