**CSC 338 Parallel and Distributed Computing**

**Exercise No. 3, February 6, 2017**

**Introduction to the programming environment**

**Goal**

Compare multithreading with multiprocessing for a web page download task.

**Background**

Which is better—multiprocessing or multithreading? As is often the case in computing, the answer is: it depends on the problem you’re trying to solve.

**Procedure**

Use the command line to execute programs in the following steps.

1. Open web\_fetch\_serial.py (get it from *trace).* This program downloads a number of web pages using the urllib module. Execute the program several times and note how long it usually takes.
2. Save web\_fetch\_serial.py as web\_fetch\_threads.py and modify it to use a separate thread for each download. This will require that threads coordinate so different threads don’t download the same web page. An easy way to do this is to use a Queue from the queue module: que = queue.Queue(); be sure to import queue. I suggest you make the queue global although, because Python passes collections by reference, you could define it in main and pass it to get\_page().

In main():

* 1. If you want the queue to be local, create it here. Either way, put the hosts into the queue (use the queue’s .put() method).
  2. In a for loop, spawn a thread for each host (use for i in range(len(hosts))). I suggest you use t for the name of each thread and give the thread a name such as “gp” + str(i). The target function is get\_page(); if your queue is local to main(), pass it to the thread using *args.*
  3. Start the thread but don’t join it.
  4. After the loop that creates and starts the threads, wait on the queue to be fully processed (use the queue’s .join() method).

In get\_page():

1. If the queue is local, get\_page() takes no arguments. If you’re passing the queue, then make it an argument to get\_page().
2. Get a host from the queue (use the queue’s .get() method).
3. Open the url, as in web\_fetch\_serial.py, and read the html.
4. Print the name of the thread and the first 50 characters from the html.
5. Tell the queue the task is done (use the queue’s task\_done() method).

Execute web\_fetch\_threads.py several times (you could use a loop) and note how long it usually takes. Is it faster than web\_fetch\_serial?

1. Save web\_fetch\_threads.py as web\_fetch\_mp.py. Modify it so that, rather than using threads to download pages it uses multiple processes. I suggest you use 5 processes for 10 web pages; you might later try different numbers of processes, depending on the number of cores your computer has. For this program, you don’t need a queue—you can send the urls to each process. For example, if you have 5 processes and 10 urls, you can set up a loop that steps by two and sends two urls to each process.
2. Our internet may be too fast to show any effect. Put the get\_page() function to sleep (in both programs; use time.sleep(*n*) to sleep for *n* seconds) for some length of time to simulate waiting on I/O. Does that make any difference?

Execute web\_fetch\_mp.py several times and note how long it usually takes. Is it faster than web\_fetch\_serial.py? web\_fetch\_threads.py? What have you learned?