Programming Assignment #1
Multi-Core MapReduce

CS 416: Operating System Design
Due 11:55 PM, October 9

Professor: Abhishek Bhatacharjee
TAs: Dennis Vranjesevic and Zi Yan

Group Size: 3 per group (one group of 4)
Programming Language: **C or C++ ONLY**

Overview

For this assignment, you will implement a framework for multithreaded data processing on a single computer that follows the "MapReduce" paradigm.

MapReduce is a popular programming paradigm for data intensive computing in clustered environments such as enterprise data-centers and clouds. MapReduce is used as a framework for solving number of parallel tasks, using a large number of CPUs or computers (nodes), collectively referred to as a cluster.

The name MapReduce comes from the two-phase manner in which the data is processed:

In the **map** phase, the input is divided into smaller sub-problems, and distributed to individual worker nodes. The worker node then processes the smaller problem, and passes the answer back to the framework.

In the **reduce** phase, the framework then takes the answers to all the sub-problems and combines them in a way to get the final output for the problem being solved.

Assignment

1. You are required to design and implement a MapReduce framework using C or C++ on Linux.

2. Your framework must be configurable to provide parallelism in two distinct ways:
   a. Using multiple Linux processes
   b. Using POSIX threads (pthreads)

3. Your framework should be designed in a general manner. In particular, the problem to be solved (see item 5 below) should be encapsulated solely in the implementation of the map() and reduce() functions. Substitution of different map() and reduce() functions should enable your framework to solve a different problem without ***any further changes in the framework itself.***

4. All intermediate data that is to be transferred between the map and reduce phases is to be stored in **POSIX shared memory**. Reading and writing to this shared memory may require synchronization (mutual exclusion). You may NOT write intermediate output to the file system. There also needs to be synchronization (barrier) to explicitly separate the map and reduce phases.

5. Once your framework is working, you must implement two sets of map/reduce functions to solve the following problems:
   a. **Word count:** given an input file, you must count the number of times each individual word appears in the input, and output a file containing a list of words followed by their counts.
   b. **Integer sort:** given an input file containing a list of integers, you must output a file containing the same integers in sorted, ascending order.

6. You will be provided with sample input and output files for each problem. Your implementation must be able to process the example input and must produce output in **exactly** the same format as the example output file. (These sample files will be attached to the assignment on Sakai).

7. Your framework must compile to a single executable file called **mapred**, that conforms to the following command line structure:

   mapred --app [wordcount, sort] --impl [procs, threads] --maps num_maps --reduces num_reduces --input infile --output outfile

   You should be able to execute this four different ways:
   - wordcount using processes
   - wordcount using threads
   - sort using processes
   - sort using threads

   with any arbitrary number of maps and reduces.

Honors/Extra Credit

All students in the honors section must meet these additional requirements for full credit; students in the regular sections may earn up to 20% extra credit by meeting these requirements ***provided that they also meet all of the base requirements.***

1. You must implement an additional multithreaded solution for each of the specified problems (integer sort and wordcount) using a non-MapReduce parallel implementation.

2. Your additional implementation must use pthreads.

3. The additional solution must be activated by adding an additional option to the --impl flag called "extra". In this case, the --maps and --reduces arguments will not be necessary, and should be replaced by a single --numthreads [thread_count] argument.

   For Example:
   mapred --app wordcount --impl extra --numthreads 8 --input infile --output outfile

Report

You must also provide a report that contains, at the minimum:

1. The names of the group members.
2. An architectural description of your framework.
3. An evaluation of the performance difference between the process-based and thread-based implementations of your framework.
4. A description of any difficulties that you encountered during this project.

The report must be in PDF format to receive credit, **no exceptions.**

Submission

Solutions must be submitted to Sakai, attached to the assignment. Each group should nominate one member to submit their solution. Each student SHOULD NOT submit an individual copy of the solution.

Each submission must consist of two files:

1. A .tar file containing the entire contents of your git repository, which will include your source code and a Makefile. The name of the file that you will submit will be called: **pa1.tar.gz**
2. A PDF file containing the report. Do not include this inside the .tar file.

You will receive no credit for the code if it does not compile on I-Lab machines.

References

- MapReduce:
  - http://en.wikipedia.org/wiki/MapReduce
- Mutual Exclusion and Synchronization
  - http://en.wikipedia.org/wiki/Mutual_exclusion
  - http://en.wikipedia.org/wiki/Semaphore_(programming)
  - https://www.geeksforgeeks.org/mutex-lock-for-linux-thread-synchronization/
- Creating and using multiple processes
  - http://www.yolinux.com/TUTORIALS/ForkExecProcesses.html
- Creating and using multiple threads
  - http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html
- Using POSIX shared memory
  - http://www.cs.cf.ac.uk/Dave/C/node27.html
- Textbook
  - Operating System Concepts, Silberschatz and Galvin
- Creating .tar files in Linux
  - https://www.howtogeek.com/248780/how-to-compress-and-extract-files-using-the-tar-command-on-linux/