Linux Interprocess Communications

Summary Report by David Dobor

This project consists of three parts. Each part explores a single method (of many) in which processes on a Linux system can communicate. In each part the same task is performed: given a text file, output a list showing the count of each unique word (case insensitive). In each part, the *parent* process creates a *child* process. The parent process opens the text file, reads the bytes, and passes them to the child. The parent process cannot count words; the 'child' process cannot open the file, but it can count the words.

Summary of the different IPC methods provided by Linux

Several differences between the various `IPC` methods are noted. `pipes` and `message queues` involve read and write operations where the data sent by the parent is manipulated by just one child process. The control flow between the child and the parent is automatically *handled by the kernel.* The `shared memory` model, on the other hand, allows the parent process to make data available simultaneously to several other processes that share the same region in memory. This model may fit well with designs that need to share a data structure or maintain some shared state. However, there is a need to *handle synchronization* in the 'shared memory' case, which may add to the complexity of this model.

Because `message queues` and `pipes` rely on the kernel for control flow and synchronization, they tend to be slower than the `shared memory`. We try to verify this empirically by timing our programs.

On MapReduce and Hadoop

Map Reduce refers to two separate tasks of processing (usually large amounts of) data. The tasks are `map` and `reduce`, and are performed sequentially. The map job is the first task in the sequence. It takes some data as input and converts it to another form where individual elements are key-value pairs (tuples). There are usually several mappers working in *parallel* to produce this output of tuples. The Reduce job then takes the output of the mappers and combines them to produce a smaller output of its own. This output is also a set of key-value pairs. An example of this is the third part of the project where 4 mappers processed a quarter of a large text file. Each mapper outputs the words found in its chunk of the text, together with the frequencies with which these word occurred. The reducer then combined these results to produce the frequencies of unique words in the original text file.

Hadoop is a popular software framework for storing massive amounts of data and for fast concurrent processing that data. Among the many reasons for Hadoop?s popularity are its open source nature, its massive storage capability, and its enormous processing power. Hadoop also offers advantages over traditional databases that require preprocessing of data before storing it; Hadoop users can store virtually limitless amounts of any kind of data and then decide how to use it later. An increase in demand for fast processing of various types of data - especially from social media, the internet of things, or the financial industry - coupled with Hadoop's ability to store *any* kind of data, and *process it in parallel*, quickly, using clusters of low-cost computers, is sure to contribute to Hadoop's popularity in the future as well.