Operating System Design

**Assignment # 1**

Multi-Core MapReduce

Due date: October 4, 2013

Professor: Ricardo Bianchini

TA: William Katsak

Group Members (ID#)

Taichi Hirao (134005387)

Michael Spaniol (133002726)

Theophilus Mensah (129006665)

**Objective**

The assignment was to implement a framework for multi-threaded data processing on a single computer that follows the “MapReduce” paradigm. In the map phase of the implementation, the framework takes in an input, divides it into subproblems, and distributes the subproblems among different processes/threads. The processes/threads process their inputs, and returns its output to the framework. In the reduce phase of the implementation, the framework collects all the outputs of the map phase and combines them to form output to the main problems that is being solved. This framework was then extended to solve a file word count problem, as well as a file integer sort problem.

**Architectural Description of Framework**

The framework attempts to solve the “Mapreduce” problem by first taking in the number of process/threads, the number of reducers, the input, and output filenames from the command line arguments. The framework then creates a global variable for the input and output files, as well as the job number for each process/thread to be used to access the task assigned to it. The framework then checks the size of the file, and divides it among the number of processes/threads. This division of task according to the size and number of mapper ratio could lead to wrong outputs, so the task distribution was optimized to find the nearest end of line after the gross division of the size of the file among the number of processes/threads. The task distribution was also stored in a global array for easy sharing.

In the map phase, each map process/thread gets a task with its job number and processes it. It scans its specified segment of the file and stores it in a linked list in a “word-count” format structure. The words in the file are scanned with no duplication, ie each word appears only ones in the list, and its count is increased when it is repeated. The result of each process/thread is then stored in shared memory to be used in the reduce phase. The shared memory is locked to avoid race conditions.

In the reduce phase, the number of maps from the map phase is divided among the number of reducers. Each reducer combines the different maps that it is assigned into one list. The final list of each reducer is then added to the master list and stored in shared memory to be sorted and written to file by the framework. The framework uses “mergesort” to sort the master list and write the content of the final sorted list to the output file.

**Performance Evaluation**

This implementation of the map reduce framework isn't very memory intense as it is only doing sorting. For this reason, threads have better performance than processes. Creating a process requires the system to duplicate the parent process so it has to spend time making those copies. This is very useful for larger programs that use much larger databases that also need high reliability, but this implementation is small. Threads split the process into parts and assign each part gets assigned to a core. They share the same memory space and can communicate to each other directly so the run time was faster for this program.

**Conclusion**

The map reduce program was very intuitive, but very complex to design. The design took 50% of the total time in building the program. The idea of using multiple threads/processes was very useful in speeding up the performance of the program. Although the thread implementation was more efficient than the process implementation, the benefit of using multi-threaded data parallelization was brought to light. The main challenge in this assignment was team coordination.