CS/SE 4348: Operating Systems
Programming Assignment 2

Instructor: Ravi Prakash

Assigned on: July 5, 2016
Due date: July 18, 2016

This is an individual project and sharing of code is strictly prohibited and will be dealt with as per the university's rules governing academic misconduct. You are expected to demonstrate the operation of your project to the instructor or the TA. You can choose C, C++ or Java as programming language for this project. However, all instructions below use the syntax of C and a POSIX-compliant operating system.

Map-Reduce

Introduction

The *Map-Reduce* model is inspired by the *map* and *reduce* functions commonly used in functional programming languages like *ML*, *Haskell*, *OCaml* etc. *Map-Reduce* is a framework for processing parallelizable problems across large datasets using a large number of computers (clusters). In an abstract view, it consists of two major phases. In the *map* phase, each cluster performs a local process (e.g. filtering and sorting of a partial dataset) and in the *reduce* phase a summary of the data processed in the *map* phase is collected.

Project Description: Simple Map-Reduce

In this assignment you have to implement a simplified Map-Reduce problem. For the sake of simplicity, in this project we consider several mappers and a single reducer, even though in practice there are more actors contributing towards maximizing the parallelism. Assume that we have a large text file and we want to count the frequency of a *keyword* in the file. Since, sequential search by a single process is too time consuming, we have decided to do it in *Map-Reduce* fashion. Hence, the main process (which also, subsequently, acts as the reducer) assigns mutually disjoint sections of the file among several child processes (mappers). Each mapper steps through its part of the file, counts the frequency of the *keyword* and returns the count to the main process (reducer). The reducer sums up all the partial frequencies received from the child processes.

Implementation guidelines

1. Number of mappers, search keyword and the physical name/path of the large text file are received from the command-line (argv).

2. The main process initiates the child processes by calling *fork()* for as many times as specified by the user. Be careful of unwanted identical *fork()* calls in the children processes. Hint: Use the *child process id* to distinguish the cases.

3. The processes use pipes for inter-process communication. There is no restriction on the number of pipes you use. However, you should justify their use.

4. The main process, based on the file-size and the number of mappers entered by the user, determines the start and end offsets for every child process and passes it through the pipes once the child process is created.

Data Collection

The main process, upon exit, shows a summary of the search including the number of mappers, the starting and ending offset between which the search is performed by each mapper and the frequency of the *keyword* found by each mapper, as well as the total frequency of the *keyword* in the file.

Submission Information

The submission should be through eLearning in the form of an archive consisting of:

1. File(s) containing the source code.
2. The makefile.

Please do "make clean" before submitting the contents of your directory. This will remove the executable and object code that is not needed for submission.

Your source code must have the following, otherwise you will lose points:

1. Proper comments indicating what is being done
2. Error checking for all function and system calls