CS342 Operating Systems - Fall 2016
Project 2: Multi-threaded Applications

Assigned: Oct 21, 2016
Due date: Nov 03, 2016, 23:55

You will do this project individually.

Objective: Practicing thread creation, multithreaded programming (by use of POSIX threads), and performing experiments and drawing conclusions.

Write a multi-threaded C program (by using Pthreads library) that will perform the following. It will be given a set of N input files (text - ascii files). N can be in range [1,10]. Each input file will represent stored call detail records of a telecom company including the following information in each line: callerID calledID year. This information in a line can be called as call record. For simplicity, assume callerID and calledID are just unsigned integers (8 digit integers in range [10000000, 99999999]) and year is an unsigned integer in range [1900, 1999]. A line (a call record) represents a call made by a person x to a person y in year z. There may be many such calls made in a given year from a person x to a person y (i.e., repeated information may exist) and the information about these calls does not have to be sitting in one input file, but may be distributed in many input files. The data in all input files is unsorted. The program will be given a year range (start and end year - inclusive) and ID range (start and end ID – inclusive) and will finally produce an output file that will include in each line the number of distinct people a callerID (a person) in the range called in that year range. The output should be sorted with respect to callerID. Each line of output will include a callerID and a count value.

Each input file will be processed by a separate mapper-thread that will read the whole file content and partition the lines (call records) into R intermediate temporary files (R can be at most 50). The partitioning rule is the following: a line (callerID, calledID, year) coming from an input file j (0<=j<=N-1) will go to an intermediate file "tempj-i" where i = CallerID mod R (i.e., a hash function is used to partition the input lines). Here 0<=i<=R-1. In this way the call records (lines) in an input file are partitioned into R files (at most) with respect to callerID information. Since each one of N mapper-threads can produce at most R files, there can be at most N*R intermediate files produced.

There will be R reducer-threads. Each reducer thread will perform sorting and counting. A reducer thread will read its respective N partitions (N intermediate files), sort the content according to callerID, find the count of unique people that each callerID called, and emit (callerID, count) pairs into a temporary output file. For example, the reducer thread 0 will read intermediate files (partitions) temp0-0, temp1-0, temp2-0, …, temp(N-1)-0. The reducer thread 3, for example, will read the files temp0-3, temp1-3, …, temp(N-1)-3. A reducer thread may emit (callerID, count) pairs as follows (one pair per line).

```
23790879   112
```

```
19067323   23
87908456   65
```

This means, for example, person 23790879   called 112 distinct people in the given year range.

At the end, R temporary output files will be produced. Then a merger thread will merge these files and will produce a single final file that callers and their call counts to unique people in sorted order with respect to callerID. The name of the final file will be taken from the command line as the last argument of the program. Name your program as `callcount`. It will be invoked as follows:

callcount <N> <R> <infile1> … <infileN> <finalfile> <syear> <eyear> <sID> <eID>

An example invocation is:

```
callcount 3 5 i1 i2 i3 out 1990 1993 54500999 54568900
```

Experiments: Do some experiments. Measure the time of execution, for example, for various input sizes, input data, N and R values. Draw diagrams, tables, or charts. Try to explain the results and try to draw some conclusions. Try to think and design some other experiments. Put all into a 2-3 page report.

Submission: Submit through Moodle. Put your callcount.c file, Makefile, report.pdf file, and README file into a project directory, as in project 1 (use the same naming and foldering conventions); tar and gzip the directory; and upload, as in project 1.

Additonal Information and Clarifications:

- A project skelaton is in github: https://github.com/korpeoglu/cs342-fall2016-p2.git
- More clarifications can be put in course webpage.