CS342 Operating Systems – Spring 2020
Project 2: Threads and Synchronization

Assigned: Mar 15, 2020 Document Version: 1.1
Due date: April 3, 2020. Time: 23:55

You will do this project individually. You have to program in C and Linux. We will do our tests in the following distribution of Linux: **Ubuntu 18 LTS – 64 bit**.

In this project you will do the Part C of Project 1 again, but this time as follows. Mapper threads and reducer thread will run concurrently, and a mapper thread will just read lines from its input file (split file) and will pass them to the reducer thread. The reducer thread will do the multiplication of the value in a line with the respective vector element. There will be a buffer (implemented as an array or linked list) between a mapper and reducer. If there are *K* mappers, then there will be *K* such buffers. Each mapper will put its line to its buffer. Reducer will retrieve lines from the buffers whenever lines are available. If there is no line in any one of of the buffers, reducer can wait (synchronization required). Similarly, if the buffer of a mapper is full, the mapper will wait (synchronization required).

The name of the program will be **mvt_s** and it will be invoked as follows.

mvt_s *matrixfile vectorfile resultfile K B*

B is the size of the buffer between a mapper and reducer. *B* can be a value between 100 and 10000.

To remind, the application will multiply an *n*x*n* matrix **M** with a vector **v** of size *n*. For matrix **M**, the element in row *i* and column *j* is denoted as m_{ij}. For vector **v**, *j*th element is denoted as v_j. Then the matrix-vector product is the vector **x** of length *n*, whose *i*th element x_i is given by:

$$x_i = \sum_{j=1}^{n} m_{ij} v_j$$

The matrix **M** information is stored in an input text file (ascii). The vector **v** information is also stored in an input text file (ascii). We assume that the row-column coordinates of each matrix element is also is stored in the file. Hence for each non-zero matrix value we store a triple (i, j, m_{ij}) in a line of the *matrixfile*. The line format is:

<rownumber> <columnumber> <value>

The main thread will read the *matrixfile* (which can be quite large) and will partition it into *K* splits (each split will be a file), as in project 1. The main thread will also read the whole *vectorfile* and put the vector values into an (global) array in main memory. If the vector is of size *n*, the array size will be *n*.

After generating the split files, the main thread will create *K* mapper threads and 1 reducer thread. Each mapper thread will process another split file. Mappers will run and process their splits concurrently.

The processing of a split file in a mapper thread will be very simple in this case. A mapper will read its split file one value at a time (one line at a time). Each line read will be put into the respective buffer. If buffer is full, the worker will wait (synchronization required). While putting a line into the buffer, we need to make sure that no race condition happens (i.e., synchronization required). Each line of the split file will be read and processed as described. After a mapper has finished with processing its split file, it can indicate this with a special value (line) to the reducer (or you can use some other method you can think of).

Reducer thread will run concurrently with the workers. The reducer thread will retrieve lines from buffers and will processes them (retrieve one line, process it, retrieve another line, process it, ...). Processing a line includes multiplying with the respective vector element and adding the multiplication result to the accumulating sum in the respective entry of the Result Vector; as in project 1). In this way, reducer thread will generate a result vector. When all processing is done, result vector will be written to the result file as in project 1.

Submission

Put all your files into a project directory named with your ID (one of the IDs of team members), tar the directory (using **tar xvf**), zip it (using **gzip**) and upload it to Moodle. For example, a student with ID 20140013 will create a directory named 20140013, will put the files there, tar and gzip the directory and upload the file. The uploaded file will be 20140013.tar.gz. Include a **README.txt** file as part of your upload. It will have the names and IDs of group members, at least. Include also a **Makefile** to compile your programs. We want to type just **make** and obtain the executables. Do not forget to put your report (PDF form) into your project directory.

Additional Information and Clarifications

- Again the project is to teach threads, synchronization and semaphores. We would normally not write this program the way described for a single compute node.