CS342 Operating Systems, Spring 2012

Project 2: Synchronization

Assigned: 21.03.2012
Due date: 04.04.2012

> *"...The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise. --- Edsger Dijkstra*
>
> *"You know you've achieved perfection in design, not when you have nothing more to add, but when you have nothing more to take away". ---Antoine de Saint-Exupery*

In this project, you will implement a multi-threaded program, called `merge`, that will merge N sorted input files (containing integers) into an output file that will also be sorted. The integers are sorted in ascending order. The program will take the following command line parameters:

merge N outfile file1 file2 file3 …. fileN

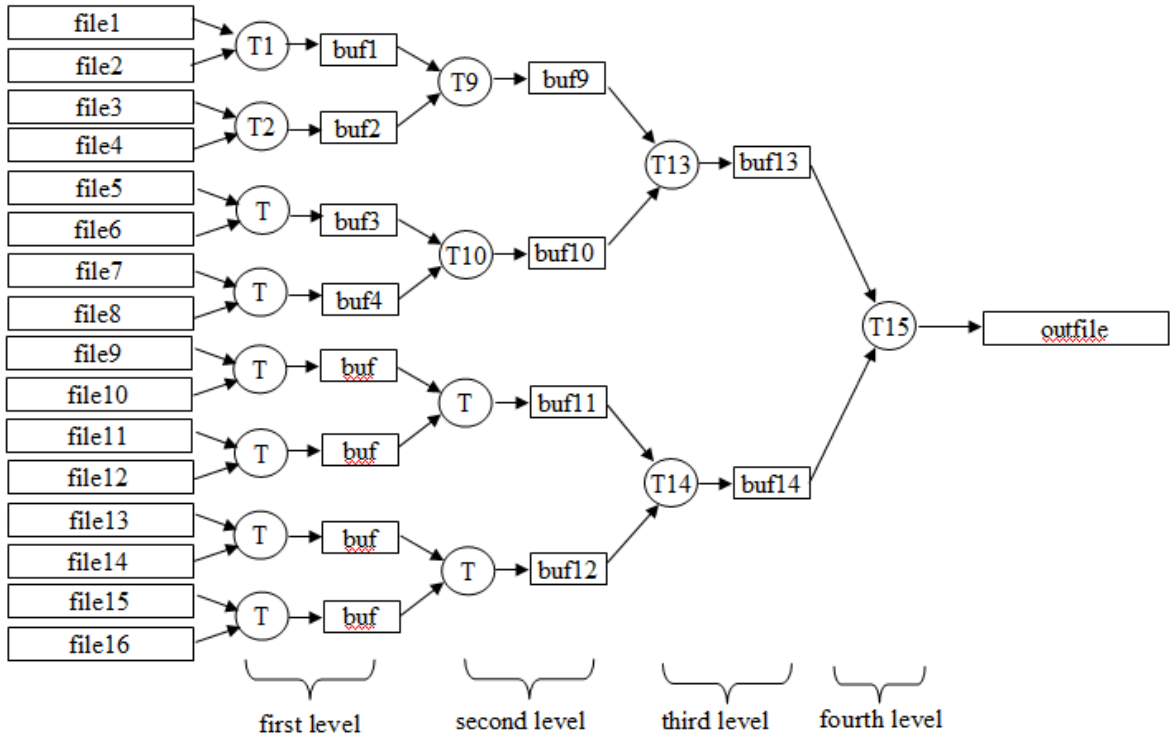
where:
- N is the number of input files (N will be power of 2).
- outfile is the name of the output file.
- file2 file3 …. fileN are the names of input files.

Your program should be implemented in the following way. For each input file pair, it will create a thread to merge those input files. The output (sequence of integers) will go into a bounded buffer (array). Lets us call such a thread as a first level thread and such a buffer as a first level buffer. For N input files, there will be N/2 such first level threads and first level buffers. There will be also second level, third level, … threads and buffers created. A second level thread will merge the integers arriving through two first level buffers. The output of the second level thread will go into a second level buffer. Similarly, a third level thread will merge integers arriving through two second level buffers into a third level buffer. This will go on. At the highest level, there will be a single thread merging two buffers into the output file.

Figure below shows a typical run-time structure of the program for the case where N is 16. As we can see there are 8 first level threads and 8 first level buffers used. There are 4 second level threads/buffers, 2 third level threads/buffers and 1 forth level thread.

Merging two input sequences of sorted integers into an output sequence of sorted integers can be performed efficiently. We just look to the head of the sequences (i.e., to the integers at the head of the queues/files). We compare those two integers and remove the one that is smaller and write it to the output sequence (buffer/file). If there is a tie (both heads are equal), we can remove both of the integers and put/write them into the output sequence (i.e., buffer/file). We continue doing this, i.e., again look to the heads of the two input sequences and remove the smaller integer. When one of the sequences is finished, we just take integers from the other sequence and finish that as well. To give an example lets us consider the following two sequences: S1: 3 7 9; and S2: 4 7 8 8 10

Head of S1 is 3 and head of S2 is 4. We compare them and we remove 3 from S1. Then head of S1 becomes 7. We compare 7 (from S1) with 4 (from S2) again and we remove 4 from S2 this time. If we continue this, the order of access to these sequences will be: 1, 2, 1, 2, 2, 2, 1, 2.



You will use **Pthreads** to create and use threads. You can use **POSIX Pthreads semaphores**, or **Pthreads mutex/condition** variables or all of them to implement synchronization requirements.

Some synchronization requirements are:

- Access to buffer should be mutually-exclusive.
- When both of the input buffers are empty, a thread will go to sleep.
- When output buffer is full, a thread will go to sleep until an item is removed.

Notes:

- Max value of N is 16. Min value is 2. N is power of 2.
- An input file (ascii text file) will contain a sequence of integers in sorted order. Integers may not be unique. Additionally, there may be negative integers in the input file at any place. You will read those integers from the file but you will ignore them. In an input file, the non-negative integers should be in sorted order; but the negative integers does not have to be in sorted order obviously.
- The output file (ascii text file) will contain only non-negative integers.
- Each line of input file may contain zero or more integers. If you use fscanf(fp, "%d", ..), you can easily read the next integer no matter where it is in the file. If the integer you read is negative, just ignore it and read the next integer.
- The output file will contain exactly one integer in each line.

The following can be a sample input and output:

File 1: 3 7 -1 -1 10 15 -4 20 -2 -2 -2 -2 -3 -3 -3 25
File 2: 0 -1 -1 1 2 -5 7
Sorted out file: 0 1 2 3 7 7 10 15 20 25 (each integer in a separate line)

Submission

You will submit your program, merge.c, together with a Makefile.