COP-4338 Systems Programming

Programming Assignment 5

Knight Foundation School of Computing and Information Sciences

In this assignment, you are asked to implement a multi-threaded word search puzzle solver that uses low-level I/O to read the file containing the puzzle table.

The puzzle table is assumed to be a square-shape 2D array of characters with n rows and columns where $15 \leq n \leq 46340$ and n is received from the command-line.

To solve the puzzle, your program needs to use a hash set keeping the same dictionary that was used for the previous assignment. The program scans the whole table to find meaningful English words whose letters appear in adjacent cells located in:

- the same row of table from left to right or from right to left,
- the same column of table from top to bottom or from bottom to top,
- a line of slope $+1$ from left to right or from right to left,
- a line of slope -1 from left to right or from right to left.

The technique used to solve the puzzle using a multi-threaded program is Divide & Conquer: (1) The given square-shape 2D array is divided into multiple square-shape 2D sub-arrays. (2) The solution of the puzzle is the aggregation of solutions to all the sub-puzzles generated in step 1.

The program uses the producer/consumer model.

- producer: reads a square-shape sub-array of the given 2D array from the input file using low-level I/O system calls (open, read, lseek). There is no need to have multiple producers as there is a single input file. Therefore, you may use the main thread itself to play the role of producer.

- consumer: searches for meaningful words in the hash table containing the dictionary. The main thread may spawn multiple consumer threads to enhance the search speed using concurrency.

Also, assume that the program allocates 64MB (million bytes) memory for the buffer used by producer and consumers to exchange data. Number of equal-size cells in this buffer is passed by the command-line argument and can be one of the following values: 1, 4, 16, and 64.

Command-Line Arguments and Options

Your program may be executed with the following command-line arguments (options):

- **-input yyy**: where yyy is the name (and possibly path and extension) of file containing the square-shaped puzzle. To make your own random puzzle, you may use the program stored in random_puzzle_generator.c available on Canvas.

- **-nbuffer zz**: where zz is the number of equal-size buffer cells that can be 1, 4, 16 or 64. If the number of buffers zz does not satisfy the mentioned condition, your program must print an error and exits immediately. Your program must spawn a consumer thread for each of the non-empty buffer cells so that the puzzle-solving process happens concurrently. Therefore, the number of buffers also specifies the maximum number of concurrent consumers who solve the puzzle.

- **-dict ddd**: where ddd is the name (and possibly path and extension) of file containing the dictionary. The file dict.txt which contains the list of all meaningful English words is available on Canvas.

- **-size ss**: where ss is the number of rows/columns of the square-shape puzzle table. It can be any number from 15 to 46340 (inclusive). If the size ss does not satisfy the mentioned condition, your program must print an error and exits immediately.

- **-len a:b**: where a and b are positive integers specifying minimum and maximum length of words that solve the puzzle. Please note that $3 \leq a \leq b \leq 14$. If a and b do not satisfy the mentioned condition, your program must print an error and exits immediately.

- **-s**: this flag means that your program must print the list of words that solve the puzzle in alphabetical order. Implementation of this option requires the usage of mutex locks to synchronize access to a collection object kept externally. You can use a binary search tree to collect words so that you can easily print them all at the end.

Submissions

You need to submit a *.zip* file compressing the C source file(s) related to the assignment (*.c* files), header files (*.h* files) and the makefile. **Please use "solve" to name the executable of your program in the makefile**.