DESIGN OVERVIEW DOCUMENT

PROBLEM STATEMENT:

Design a simulation of a line scanner currently with 2 parallel blocks namely-

1. Data Acquisition/Generation Block- This block generates two consecutive elements for every time interval T or acquires two consecutive elements from the test file.
2. Filter and Threshold Block- This block receives input from the Data Acquisition/Generation block and flags the input pixel as 1 or 0 based on the criteria and threshold value.

NAMING CONVENTION:

1. Classes/Methods/Functions- UpperCamelCase.
2. Variables- lower_case_underscores.
3. Macros/const static variable- UPPER_CASE_UNDERSCORES.

DEFINITIONS:

1. ***PIXEL_RANGE***- Intensity of the pixel. In this case it is taken as 8-bit integer (0-255).
2. ***WINDOW_SIZE***- Size of the filter window, it is given as 9 in problem statement.
3. ***PIXELS_TO_BE_SCANNED***- Number of pixels scanned at a time by the line scanner.
   It's value is given as 2 in the problem statement.
   Assumption- At the end of the row, if number of pixels left is less than PIXELS_TO_BE_SCANNED then those remaining pixels are scanned.
4. ***TEST_FILE_PATH***- Path of the test file.
5. ***Inputs Given By User-***
   1. number_columns- Number of columns in each row(m)
   2. threshold_value- Threshold value to flag the pixels.
   3. process_time- Iteration time T in nanoseconds.

      Assumption- Line scanner takes input for every time interval T.
   4. input_mode- `Data Generation mode or Data acquisition from test file mode.`

SOLUTION:

The problem can be viewed as a modification to producer-consumer problem. The producer here is the Data Acquisition/Generation Block and consumer is the Filter and Threshold block. Here the producer produces until the buffer size is less than WINDOW_SIZE and consumer consumes only if buffer size is greater than or equal to WINDOW_SIZE.

Inter-process communication used by the two process blocks here is ***shared memory*.** The two process blocks share the object input_buffer. The input_buffer object has methods

1. ScannerFeed()- This method fills the buffer_queue by the pixels generated by Data Acquisition/Generation block until it's size is less than WINDOW_SIZE
2. Filter()- This method processes the buffer_queue pixels according to the given criteria and flags the pixel as 0/1 according to the threshold value given by user.

Data Acquisition/Generation Block- This block operates in two modes according to the input given by the user-

1. MODE_GENERATING_DATA- In this mode, this block generates random pixels using rand() function and fills the vector generated_pixels until it's size is less than PIXELS_TO_BE_SCANNED or if generated pixels in the row is equal to number of columns. After generating the required pixels, the vector is passed to ScannerFeed method to fill the input_buffer.
2. MODE_TEST_DATA- In this mode, the block reads PIXELS_TO_BE_SCANNED (or less in border case) number of pixels from the FILE_PATH provided and fills the generated_pixels vector and then the vector is passed to ScannerFeed method to fill input_buffer.

Filter and Threshold block- This block calls the filter method of input_buffer object.

For now, I am printing the flag value of the pixel (0/1) to the output, if another block uses this output it can be stored in another buffer queue and processed in parallel.

A Thread is created for each process and these threads run parallel.

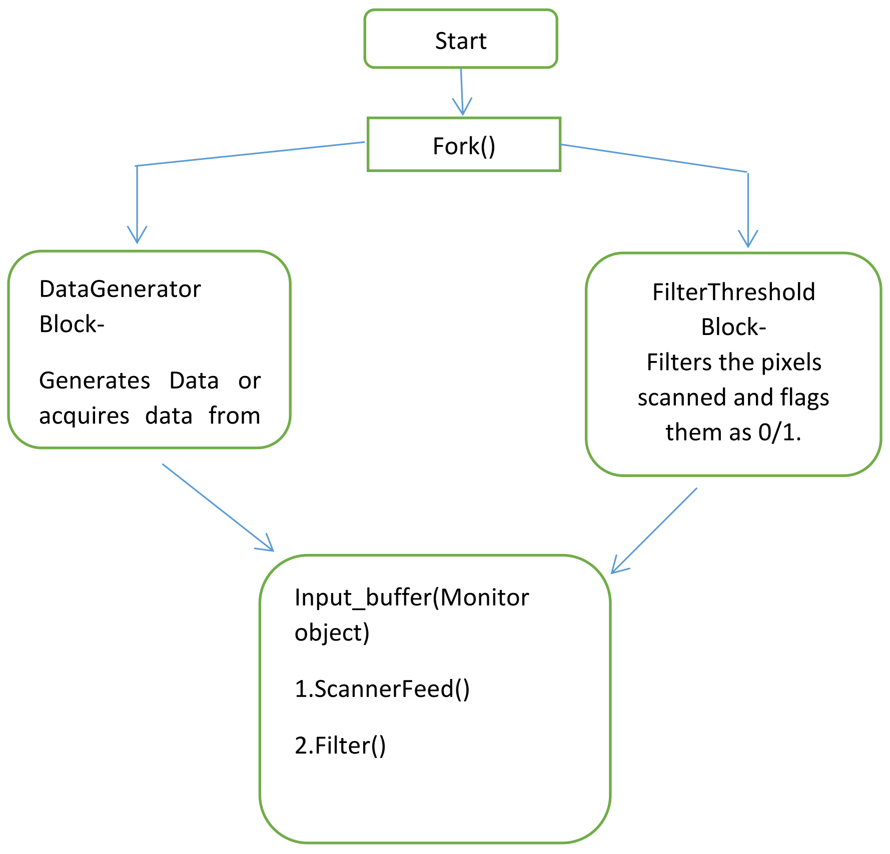
Both these process blocks are synchronized using the below Design pattern.

DESIGN PATTERN:

Concurrent design pattern used for this problem is ***Monitor object pattern.*** The monitor object guarantees that only one method runs within an object at any given point of time.

In this the monitor object is input_buffer, by using mutex lock and condition variable, it is made sure that only one method (i.e. either ScannerFeed or Filter method) runs at any given point of time.

This design pattern ensures both modularity and scalability of the code.


Start
Fork()
DataGenerator Block-
Generates Data or acquires data from
FilterThreshold Block-
Filters the pixels scanned and flags them as 0/1.
Input_buffer(Monitor object)
1.ScannerFeed()
2.Filter()

IMPROVEMENTS:

1. Configuration file can be maintained to get PIXEL_RANGE, TEST_FILE_PATH, etc… instead of hard-coding them, configuration file can be parsed to get these parameters.
2. Graphical representation of input pixels and filtered values of these pixels.
3. We can get the entire row and filtered values of different windows can be computed in parallel. (similar to vectorization)

Above improvements are not made due lack of time.

Instructions to run the code:

1. Run .\Line_scanner\x64\Debug\Line_scanner.exe file.
2. You will be prompted to enter the number of columns, Threshold value, process time and mode of input.

   Note: Number of columns must be greater than WINDOW_SIZE.
3. Also, for MODE_TEST_DATA, I have provided two test files.

   1)test_large.csv (720*1280) 2)test_small.csv(8*12)
4. By default, test_large.csv is set.
5. To change it, modify the TEST_FILE_PATH macro in utils.h header and rebuild the solution.
6. After you enter the mode, the console will display the pixel, its filtered value and flag in the following format-

   ->Pixel:

   ->Filtered value:

   ->Flag:
7. All the dependencies libraries(.dll) are provided in dependencies folder.
8. In case of any error, copy the dll files in the directory in which Line_scanner.exe is present.

Language: C++14

IDE: Visual Studio 2019

OS: Windows