MONITORS – THE NEW TEXT EDITOR

1 OBJECTIVES

The main goals of this assignment are to learn and implement the follow concepts of multithreading:

- Monitors as synchronization mechanisms.
- Bounded buffer as shared resource for communication between multiple threads.
- Writer/Reader pattern.

2 DESCRIPTION

Create an application that reads the contents of a file from a source, such as a hard disk, into memory. Allow the user to manipulate the text by searching and replacing words, and display the edited text using a component on the GUI. As an optional feature, the edited text can be saved to a destination file.

To program this assignment, apply a Writer-Reader model. To achieve this, the actions of reading, modifying, and writing must take place concurrently using threads. The text editor should use multiple writer threads, multiple modifier threads, and a single reader, although multiple readers are also allowed. Threads communicate using a shared buffer, which can be an array of strings with a limited capacity, such as 20. Alternatively, you may use a dynamic collection with a maximum capacity accordingly.

Multiple Writer threads should perform the action of writing, with each thread picking up a string from the source. The source is a list of strings loaded from a file and each thread should store its string in the shared buffer. Multiple Modifier threads should each collect a string from the shared buffer, modify its content, and save it back in the same position of the shared buffer. The Modifier thread that processes a string should mark it for the Reader thread.

The application should be GUI-based and can be programmed in either C# or Java. The GUIs for both languages will be provided. However, if you prefer to create a console application, you may do so, but it may require more work to allow the user to select a text from the source and provide a replacement.

To clarify, a console application will still require the user to input the file path and replacement text via the console, rather than through a GUI interface. This may be less user-friendly than a GUI-based application. Additionally, a console application may require more manual error handling and input validation compared to a GUI-based application.

3 TO DO

The user can select a text file from a source via the File menu or a button on the GUI. The application loads the file and displays its contents in a component (RichTextBox in C# and JTextArea in Java) on the GUI. The user can then select to replace and modify the text. The

Find and Replace feature is administered by the program through two input boxes, as in the suggested GUI image above. The user inputs a search string and the string to replace the search string. The program finds all occurrences and performs the replacement when the user clicks the button for replacement.

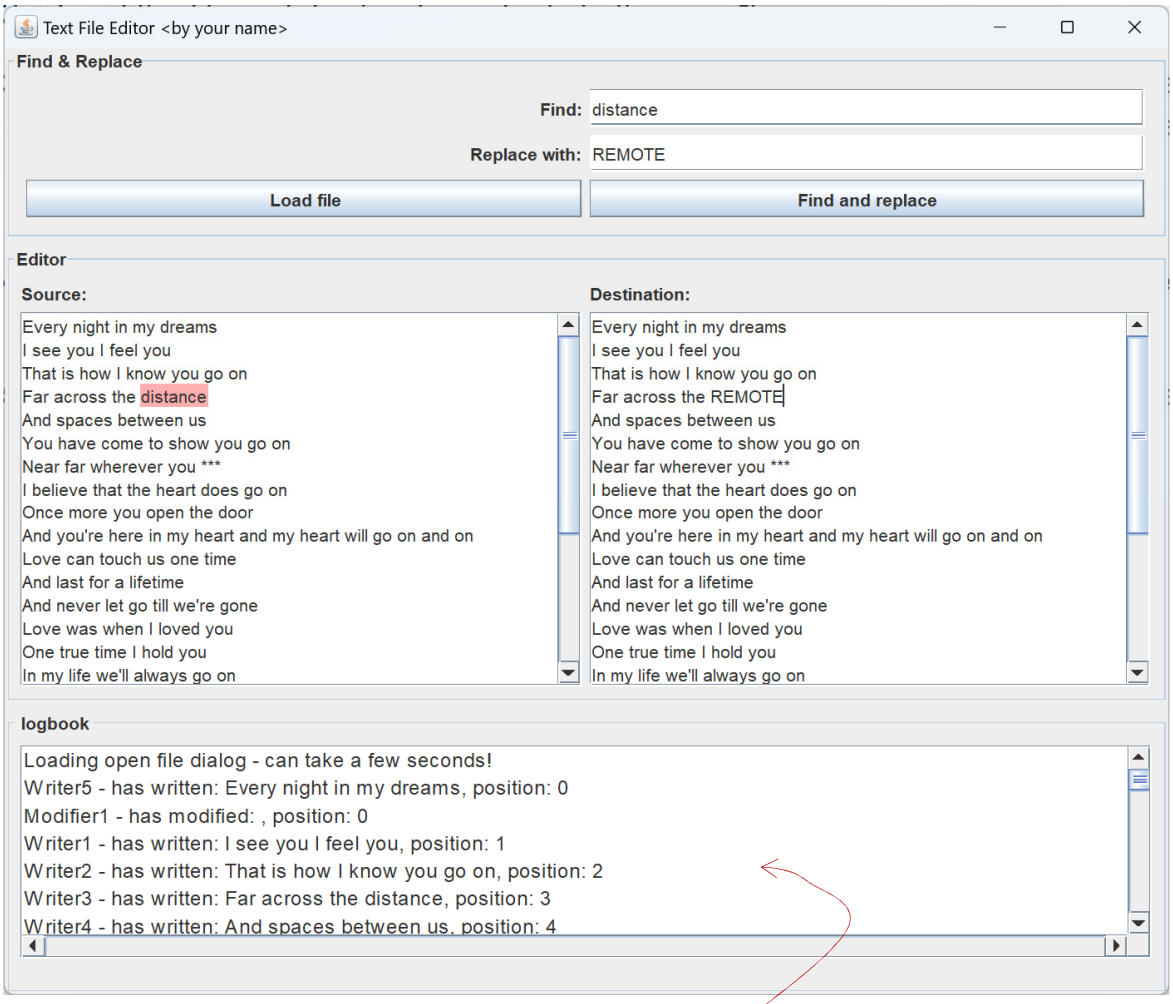
3.1 Prepare a text file with at least 10 lines of text that you can copy from any source, such as the lyrics of a song from the internet, and save it to your application directory.

3.2 Load the file and display its contents to the user.

3.3 Create at least 3 Writer threads, 4 Modifier threads, and one Reader thread. Start the threads to perform their duties. A Writer threads should not write a string that has already been written, and a Modifier thread should not modify a string that has already been modified by another modifier.

3.4 Allow the user to select a text and provide a replacement text using Find and Replace feature.

3.5 Display the edited text using a component on the GUI. Optionally, provide a feature to save the edited text to a destination file.



Log of how thread work

4 REQUIREMENTS

4.1 The synchronization of threads must be achieved using monitors and condition variables, such as **wait**, **notify**, and **notifyAll** in Java, or **Wait**, **Pulse**, and **PulseAll** in C#. This ensures that only one thread can access the shared string buffer at a time..

4.2 To manage the shared buffer, a bounded buffer class should be implemented with a maximum capacity, for example, 20. To simplify the synchronization process, each position in the buffer should be labelled with a constant, such as "**Empty**", "**New**", and "**Checked**". At the start, all positions in the buffer should be labelled as "**Empty**".

4.3 Create an enum **Status**, so you can then use an array of this enum to save the status of each element. All elements have the status **Empty** at start.

```
public enum BufferStatus
{
    Empty,
    Checked,
    New
}
```

4.4 The writer thread can only write to a location that is marked as "**Empty**". After writing, the thread set the status of the position to **New**. The modifier thread should then check the written string for a find-and-replace operation of the given substring at a position where the saved element (string) contains the search string. If found, it should make the modification at that position and change the status of this position to "**Checked**"

4.5 The reader thread should only read a position that has the status **Checked**. After reading the string from that position, it should reset the status of that position to **Empty**, allowing the writer thread to proceed and write to that position. The buffer should also have three pointers (int) to keep track of the positions in the buffer: a write pointer (writePos), a read pointer (readPos), and a modify pointer (findPos).

4.6 The first writer thread should write a string to the element at position **writePos** and update the status of that position in the status array to **New**. It should then advance the write pointer to the next position using the algorithm **(pos+1) % bufferSize**. The modifier thread can then modify this location while the writer continues until it reaches a position that is not empty. After the modifier has completed its task at the current position, it should change the status of that position to Checked and advance its position pointer. The reader thread should then read the string from that position and store it in an output list. Finally, the status of that position should be set to Empty, allowing the writer thread to write to that position again.

5 GRADING AND SUBMISSION

Submit your solution in Canvas and show your work to your lab-supervisor as before.

Good Luck!

Farid Naisan,
Course Responsible and Instructor