CSci 4061: Introduction to Operating Systems

Project 4 - Concurrency and Synchronization due: Tuesday November 23rd, 2021

Ground Rules. You may choose to complete this project in a group of up to three students. Each group should turn in **one** copy with the names of all group members on it. The code must be originally written by your group. No code from outside the course texts and slides may be used—your code cannot be copied or derived from the Web, from past offerings, other students, programmer friends, etc. All submissions must compile and run on any CSE Labs machine located in KH 1-250. A zip file should be submitted through Canvas by 11:59pm on Tuesday, November 23rd, 2021.
Note: Do not publicize this assignment or your answer to the Internet, e.g., public GitHub repo.

Objectives: The focus of this project is to use mutex to synchronize concurrent linked list operations.

1 Concurrent insertion of nodes to a linked list

Shared list structures often have concurrent access and modification. You will implement a linked list that supports concurrent insertion and traversal operations. Write a multi-threaded program (in `linked_list.c`) that takes two command-line arguments: the name of a file and number of threads between 1 and 16. Your program should support reading the input file **line by line** but concurrently by all threads. That is, the threads must **concurrently** read the lines of the file. A thread may read any line. We will have a mechanism to verify if the lines are read concurrently by the threads. You will need to use a global line-number counter and synchronize the counter to avoid reading a line multiple times or over-reading the lines. Once a line is read, the thread should create a single node and insert it into a shared linked list. The node has the following structure.

```
struct linked_node {
    int seq_no;
    int line_no;
    char *content;
    struct linked_node *next;
};
```

The `line_no` and `content` fields denote the line number and the actual line from the original file. `next` points to the next node in the linked list. The nodes are inserted in the linked list based on the `line_no` field. That is, after an insertion, the linked list is sorted in ascending order based on the `line_no` field. The `seq_no` is the sequence number (e.g., 0, 1, 2, 3, ...) denoting when the node is inserted into the linked list. You will need a sequence-number counter to keep track of the sequence number. Note that, both the counter and the linked list require synchronization.

After reading and inserting the lines in the linked list, you will traverse the linked list serially and output its content. Specifically, output the `seq_no`, `line_no`, and `content` (in the same order), comma separated, to the console.

In the correct implementation, the `content` field of the output matches the input file. Further, the size of the linked list must match the number of lines in the file. Note: All counters are zero

indexed and all lines successfully read using `fgets` are valid. You can assume that a line size is limited to 256 characters and contains only ASCII text. However, do not assume a limit on the number of lines in a file.

You must minimize critical sections, and the Mutex locks should only be used to synchronize critical sections. You can use the same Mutex lock to synchronize the sequence-number counter and the linked list. However, you must use different Mutex locks for synchronizing the line-number counter (for tracking the read lines) and the linked list.

2 Deliverables

Students should upload to Canvas a zip file containing their C code, a Makefile, and a README.txt that includes the group member names, what each member contributed, any known bugs, test cases used (we will also use our own).

3 Grading Rubric

- 5% For correct README contents
- 5% Code quality such as using descriptive variable names and comments
- 10% Project confirms to all the specifications described
- 10% Comprehensive error checking of all system calls that return an error code
- 40% Correct synchronization (at least two Mutex locks) of all critical sections
- 30% Correct traversal and concurrent insertion in the linked list with multiple threads

4 Synchronization hints

Traversal and insertion are equivalent to read and write operations respectively. First, multiple threads can read the same line and can create nodes with the same content and line_no but with different seq_no. To avoid this problem, synchronize the threads to create a unique node for each line. Second, when one thread is traversing the linked list, do not modify the linked list simultaneously. Synchronizing the linked list HEAD pointer ensures the entire linked list is synchronized. Third, avoid inaccurate scenarios leading to different content and line_no but has the same seq_no. To implement this problem efficiently, write different subroutines that can read a particular line from a file, perform node traversal, and insertion. To debug concurrency issues, test your code on smaller files.

5 Example

Suppose, you are running your program with three threads ($t1, t2, t3$) on a file - foo.txt. The text contains the following four lines:
Alpha
Bravo

Charlie
Delta

By enabling multithreading and synchronization, t1, t2, and t3 might read any line. However, each line should only be read once. Let's assume that t1 reads lines 1 and 4; t3 reads line 2, and t2 reads line 3. If we further assume that nodes for the lines are inserted into the linked list in the following sequence: Alpha, Delta, Charlie, and Bravo. Then, the output should look like:
0, 0, Alpha
3, 1, Bravo
2, 2, Charlie
1, 3, Delta

Notice that the line_no, the second column, is the basis for node insertion and the content (the third column) is the same as the lines in foo.txt. However, the seq_no will vary for each run.

6 Sample Output

./tlist foo.txt 3
1, 0, Alpha
0, 1, Bravo
2, 2, Charlie
3, 3, Delta

./tlist foo.txt 3
0, 0, Alpha
2, 1, Bravo
3, 2, Charlie
1, 3, Delta

cat foo.txt
Alpha
Bravo
Charlie
Delta

Note that the sample output does not include all the test cases we would use.