Lab exercise 7

More practice on Threading

Due date

Apr 3, 2023

Academic Integrity

The following actions are strictly prohibited and violate the honor code. The minimum penalty for plagiarism is a grade of zero and a report to the Aggie honor system office.

- Uploading assignments to external websites or tutoring websites such as Chegg, Coursehero, Stackoverflow, Bartleby, etc.
- Copying code from external websites or tutoring websites such as Chegg, Coursehero, Stackoverflow, Bartleby, etc.
- Copying code from a classmate or unauthorized sources and submitting it as your work.

Keywords

Threads.

Introduction

The objective of this exercise is to help you understand how threading works with a UI application.

Expected Functionality

In this exercise, you will have 2 threads. The primary thread will be a UI data entry point where the user will continuously give the program data until they enter ‘Exit’. The second thread will be the data processing thread that will open, write to, and close a csv file. The last entry of a row should also have the new line character, while other entries in the row will have a comma after the value. Note that since this is user input, you may treat each entry like an ascii string, so they can be any length (under 256 bytes) with any type of value.

Once the user has entered ‘Exit’, the main thread will send a signal through the message queue to stop the data thread. In this way, you’ll get introduced to creating a responsive UI application.

Inputs to the program will be -c for column names and -f for the file name.

Here's an example:

```
$ ./data_entry -c "Name ID Age" -f people.csv
$ enter data> Billy
$ enter data> 1
$ enter data> 41
$ enter data> Stephen
$ enter data> 2
$ enter data> 23
$ enter data> Kimm
$ enter data> 3
$ enter data> 35
$ enter data> Exit
```

People.csv:

Name	ID	Age
Billy	1	41
Stephen	2	23
Kimm	3	35

Starter Code

You are given the file data_entry.cpp to edit as well as the makefile required to compile the code. You are expected to use getopt to parse the column names from the -c argument as well as the output file name from the -f argument.

The class BoundedBuffer will act as a medium of communication between the UI and data processor thread; the UI thread will act as a producer to the buffer, and the data thread will consume. This BoundedBuffer is identical to that of PA3, so a correctly completed BoundedBuffer in PA3 will work in this LE, and vice versa. The first message that is sent to the data thread should be the details of the file, its structure, and column names (sent from the main thread). After the first message, string values (or char vectors) will be sent from the UI thread until the user enters exit.

The main function should create two threads, one for the UI to retrieve the user data, and the other to perform the file writes. You can decide whether to write each cell to the file individually or to wait for the full line to be retrieved before performing the write. A queue should also be created to share information between threads.

ui_thread_function: This thread handler will continuously be requesting data from the user and act as a producer to your bounded buffer.

data_thread_function: This handler will act as a consumer of the buffer and properly format the bytes to write to the file. You will need to use the write_to_file function to handle sending data to the new file.

Rubric

There are three tests that check the elements of the program which you have been asked to fix and grades will be assigned as the score you see on GitHub classrooms (under the Actions tab):

1. Compilation (10 pts)
2. Correct output (33 pts)
3. Use of threads (22 pts)
4. Threads close correctly (22 pts)
5. No leaking memory (13 pts)

Getting started

1. Go to the assignment's GitHub classroom: https://classroom.github.com/a/QtHFKKLl
2. Create a repository for your project.
3. Watch the Getting Started video: https://www.youtube.com/watch?v=kzZKR0onncs