Concurrency and Paralellism

GEI 2024

Lab Assignment #2 (Concurrent Compression)

In this assignment we are going to develop a concurrent compression tool. We will start from a single threaded version. The tool compresses blocks of a given size from the input file and creates an archive file formed by compressed chunks. An archive can then be decompressed back into the original file.

The compression and decompression functions use the zlib library. In order to compile the compression tool you must have the development headers installed. Check that you have the required packages using the package manager of your distribution. E.g, in Ubuntu:

```
$ sudo apt-get install zlib1g-dev
```

The compression tool has the following modules:

- **compress**, which interfaces with the zlib library to compress/decompress data in memory.
- **chunk_archive**, which creates and reads files that store an arbitrary number of chunks of compressed data.
- **queue**, an implementation of a queue that is used to manage the input and output of the compression and decompression processes.
- **options**, that reads the options from the command line.
- **comp**, that ties the other modules together into the compression tool.

Add the following features:

Exercise 1 (**Make the queue thread-safe**) The implementation of the queue is not thread-safe. Two threads working on the queue at the same time could make it fail. Change the queue so that concurrent accesses work correctly. Check your solution by writing a small testing prototype that inserts/removes elements from multiple threads.

The queue has a limited size, which is specified when it is created. Threads should wait if they try to remove an element from an empty queue, or if they try to add an element to a full queue following the producers/consumers pattern.

Exercise 2 (**Make the compresssion concurrent**) The comp function processes the chunks from the input queue sequentially. Modify the implementation so that **opt.num_threads** threads remove and process chunks from the input queue, and insert them into the output queue.

Note that the implementation of **chunk_archive** supports adding the chunks out of order, so you don't have to make sure they are added in order to the queue.

Exercise 3 (**Make the reading and writing of chunks concurrent**) The queue has size `q->size`, which limits the number of chunks it may hold. The maximum number of bytes that the queue may hold is `q->size * chunk_size`. The compression tool tries to read the whole file before compressing, so this puts a limit to the maximum size of the files the tool can compress.

We can increase that size by reading chunks from the original file and writing to the archive file in two separate threads. Make the reading and writing of chunks concurrent with the compression.

Submission

Assignments are due on March 3. The initial version of the code is available on github classroom at https://classroom.github.com/a/RvDu8CFb.