Concurrency and Parallelism

GEI 2023

Lab 2 – Md5 Checker

Md5 is a hashing algorithm used to map arbitrary values to a fixed length interval. A common application for these kind of algorithms is to verify that some data has not been modified somehow (such as an intentional modification, or a transmission error).

In this lab we will write an md5 checker, that given a folder and a file name performs one of two operations:

1. `md5 -s dir file`, which computes the md5 hash of all the files in the folder `dir`, and writes them to `file`. Each line in `file` will have a file name and its hash.
2. `md5 -c dir file`, which given a file generated by the previous command, checks the hashes included against the files in the folder `dir`.

Your repository includes a sequential version of this program. For operation #1, the program:

1. Goes over `dir`, and inserts the name of all the files within in a queue.
2. For each file in the queue, the program computes its md5 hash and writes it to an output queue.
3. Writes a line in `file` for each element of the output queue with a file name and a hash.

For operation #2:

1. Creates a queue with all the file names and hashes in `file`.
2. For each element of that queue, computes the hash of the corresponding file in `dir`, and checks it against the hash included in `file`. If they are different, the program prints a message.

The program accepts the following options:

- `-t n`, sets the number of threads.
- `-q n`, sets the size of the queues.

Modify the program so that:

Part 1 (**Make the queue thread safe**) Modify the implementation of the queue so that multiple threads can safely insert and remove at the same time. Calls to `q_remove` and `q_insert` should wait when the queue is empty or full, respectively.

Part 2 (**Run the scan of `dir` in a separate thread**) Reading the contents of `dir` to get the list of files to hash should be done in an independent thread. Once you finish this part, setting the size of the input queue to 1 should not make the program stop.

Part 3 (**Compute the hashes in several threads**) Start the number of threads set in the options, and compute the hashes concurrently. These threads should read a file name from the input queue, compute the corresponding hash, and write it to the output queue until all the files have been processed.

Part 4 (**Write the output file in a separate thread**) Create a new thread to write the list of files and hashes in the output file. Once you finish this part, setting the size of the output queue to 1 should still write the hashes correctly.

For operation #2:

Part 5 (**Read the hashes stored in `file` in a new thread**)

Part 6 (**Compute the hashes in several threads**) Check the hashes using the number of threads set in the options. Once you finish this part, the program should work with a queue of size 1.

OpenSSL

The program uses the OpenSSL library to compute the hashes. Check that you have the header files for this library installed on your computer. If you are using a Linux distribution based on debian (such as Ubuntu or Mint) you can install them using apt:

```
sudo apt install libssl-dev
```

Submission

The submission deadline is March 8 (at any time during that day). You can create your repository for this assignment at https://classroom.github.com/a/taAjP-bV.