ITP 30002-{01, 02} Operating Systems, Spring 2023

Homework 4. `findeq`: multithreaded search of files with equal data

Shin Hong
hongshin@handong.edu

1. Introduction

In past 15 years, hard drive cost per gigabyte has been continuously declined, and today the cost is only 10% of that in 2009. Most computer users today have virtually infinite storage in their local machines and use storage space recklessly: they easily download the same file multiple times and duplicate files if these save their time to find data. Nonetheless, still we are asked to be thrifty for using cloud storage such as Dropbox and AWS S3, or high-speed storage device with limited capacity such as NVMe SSD. In using these high-cost storage, we need to prioritize space-efficiency and remove duplication of files.

In this homework, you are asked to design and construct `findeq`, a multithreaded program that finds groups of equal files. Using `findeq`, the users can count how much memory space is wasted by redundant files and identify chances of saving up storage spaces by removing redundant files. `findeq` must use multithreading to parallelize the search to find as many identical files as possible with given computation time and resource.

Each team must submit all results by 9 PM, Fri 2 June. A submission must include all resulting program artifacts together with a presentation video that explains how `findeq` is designed and implemented and analyzes the performance of `findeq`.

2. Requirements and Constraints

2.1. User interface

`findeq` takes inputs as command-line arguments. The usage of `findeq` is as follows:

```
$findeq [OPTION] DIR
```

Basically, `findeq` receives a path to a target directory `DIR`. Given target directory, all files in the directory and its subdirectories are defined as the search scope. In addition, `findeq` may receive the following options:

- `-t=NUM` creates upto `NUM` threads addition to the main thread. The give number is no more than `64`.
- `-m=NUM` ignores all files whose size is less than `NUM` bytes from the search. The default value is `1024`.
- `-o=FILE` produces to the output to `FILE`. By default, the output must be printed to the standard output.

For invalid inputs, the program must show proper error messages to the user and terminates.

`findeq` produces the output promptly when it receives the `SIGINT` signal (i.e., when user presses `CTRL+C`), or the file search terminates. `findeq` prints the list of the filepath lists such that each filepath list enumerates all relative paths of the files having the exact same content, as discovered so far. Each list must be guarded by square brackets and each element of a list must be separated by comma and newline. For example, the following is a use case of `findeq`:

```
$findeq -t=8 -m=2048 ./Files
[
  [
    ./Files/Downloads/homework.zip,
    ./Files/ds-homework.zip
    ./Files/Data/2022-1/DataStructure/homework.zip
  ],
  [
    ./Files/Data/2022-1/ch1.pdf,
    ./Files/Data/2022-1/lecturenote/ch1.pdf
  ],
  ...
]
```

In addition, `findeq` must print the search progress to standard error every 5 seconds. A search progress must show the number of files known to have at least one other identical file, and other information about the program execution.

2.2. File search and comparison

`findeq` checks all regular files in the target directory and its subdirectory recursively, while not following hard and soft links. `findeq` do not compare the equivalence of directories, and do not consider non-regular files. You can assume that no change happens to a file in the search scope while `findeq` is running.

`findeq` must determines that two regular files are identical (or equal) if and only if their sizes are the same, and the sequence of bytes of a file is the same as that of the other files (i.e., for each offset, the two files have the same value.

2.3. Multithreading

You must design the parallelization of `findeq` to maximize the concurrency of the program execution (i.e., operate all `NUM` number of threads and the main thread) over all running time. The parallelization must be implemented with the pthread library. Using the pthread synchronization primitives, the concurrent executions of all threads must be correctly coordinated to produce valid results.

2.4. Other constraints

`findeq` must not change the files in the target directory. `findeq` must not create a child process and must not employ other computation resources. In addition, `findeq` must not call other programs and services such as database servers.

You are not allowed to use libraries other than the standard C library (https://en.cppreference.com/w/c/header) and the pthread library in developing `findeq`. Your program must be successfully built and operating in the `peace` server which runs Ubuntu 16.04 and GCC 5.4.0.

* Last update: 10:00AM, 15 May 2023

3. Presentation

You are asked to create a video to present the result. Your video must not exceed 8 minutes. The followings must be found clearly in the video presentation:

- an overview of the program design and the parallelization,
- synchronizations of threads,
- demonstrations of program executions,
- analysis on how program performance changes depending on the number of used threads.

For the last item, you need to experiment with your program to measure search performances while using different number of threads. Based on the obtained experiment data, you must analyze and discuss the relation between the program performance and the number of used threads. Note that these experiments can be conducted in your local machine. Do not conduct experiments on the `peace` server because performance measurement is unreliable since other users may run various processes concurrently on the same server.

4. Submission instruction

Each team must submit the following two items via HDLMS:

- a zip file that archives source code files, a build script, a documentation on how to build and use (e.g., README.md) and all artifacts needed for running `findeq` and reproducing all results of the presentation video.
- a URL to your presentation video. Write this URL at the submission note. It is recommended to upload the video to YouTube, Google Drive or Vimeo. The video must be available with the submitted URL by the end of the semester.

The submission must be made by 9 PM, Fri 2 June (this is strict). Please make one submission for each team, and do not make duplicate submissions.