ITP 30002-{01, 02} Operating System, Spring 2021

Homework 5. `tfind`: multithreaded text search program

Due on 11 PM, 23 Apr (Wed), 2021

Shin Hong, Handong Global University

1. Overview

You are asked to design and construct `tfind`, a multithreaded program for finding all lines of given keywords from all text files under a certain directory[1]. To utilize parallelism of a multi-core processor, `tfind` creates and operates one or multiple *worker threads* to conduct text searching on the files in multiple directories concurrently. These worker threads insert and retrieve searching tasks from a *concurrent queue*. In this homework, you must use the Pthread library to implement multithreading features including thread creation, synchronization for the concurrent queue, etc.

The submission deadline is **11 PM, 23 June (Wed)**. Your submission must include all source code files and a presentation video. The presentation must include the explanation on the program design and demo of your program with different execution scenarios.

2. Design Requirements

2.1. Functionality[2]

`tfind` is a text search program that finds all text lines that contains all given keywords at the same time. A command to execute `tfind` should be given as follows:

```
$ ./tfind -t <num> <dir> [<keyword>]+
```

`tfind` receives a positive integer (i.e., <*num*>) as the number of the worker threads (no more than 16), a pathname of a directory (i.e., <*dir*>) that `tfind` should search in, and a non-empty list of keywords (i.e., one or more <*keyword*>'s up to 8) that `tfind` should search for within a single line.

For given directory path and keywords, `tfind` must check each regular text file under the directory and all its subdirectories. `pfind` must not check the contents of non-regular files and skip links in exploring subdirectories to avoid cyclic traversals. Your program must identify whether a file is regular or not by checking its metadata, use the `file` command to check if a file is text.

For each regular text file, `tfind` checks each line whether it contains all given keywords at the same time. Once `tfind` identifies such a line, it prints out the path name, the line number and the content of the line to the standard output. For readability of a search result, each line must be atomically printed, such that no interleaving happens between multiple lines.

After accomplishing all file checks, `tfind` must print the summary of the searching result to the standard output. The result should show (1) the total number of the regular text files checked, (2) the total number of found lines, and (3) the sum of all execution time for the worker threads to process given tasks. When the user raises a signal by `Ctrl+C`, `tfind` quickly terminates its execution after printing out the summary up to the moment.

[1] `tfind` is a multithreaded version of `pfind`

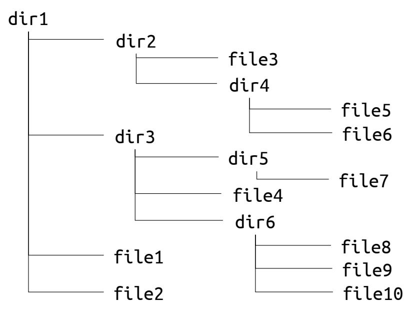

Figure 1. Search target directory example

2.2. Worker threads

`tfind` creates a given number of worker threads to assign and process *tasks* throughout a program execution. A task is identified by a pathname to the directory. A task consists of the following two operations: first, to check all regular text files to search for keyword matchings and, second, to create new tasks for every subdirectory of a given directory. For example, of Figure 1, there will be six tasks added to the task queue because there exists total five subdirectories under the search target (i.e., `dir1`).

Starting from the directory given as the command argument, all tasks are added to the task queue of `tfind`. An idle worker thread dequeues one of the stored tasks at a time, if there exists any in the queue, and then make the task done. If there is no task found in the queue, an idle worker must be blocked (not busy-waiting) until a new task is added to the queue.

2.3. Waiting queue

The task queue of `tfind` should be implemented as a waiting queue where multiple threads can enqueue or dequeue items concurrently. All operations on the task queue must be properly synchronized such that no conflict occurs even when multiple threads are invoking these operations simultaneously.

This queue should be implemented as a linked list such that worker threads can add tasks without considering a bound. As the data structure has no capacity bound, the add operation (i.e., enqueue) on this queue must not block a caller thread. In contrast, the dequeue operation of the waiting queue blocks a caller thread when the queue is empty. A blocked thread will be awoken when a new item gets inserted to the queue.

3. Implementation Requirements

Your program must use the Pthread library for implementing multithreading features. You should identify all behaviors where synchronizations such mutual exclusion and blocking are needed and implement them correctly by using suitable Pthread primitives and library function calls.

It is possible to access the list of files in a directory by using

[2] The requirement of `tfind` is a simplified version of that of `pfind`

`readdir` and determine whether a file (or directory) is regular or not (or symbolic link) by checking into the `dirent` structure. To check if a file contains text data, your program must use a Unix utility `file` to see whether the output for a given file mentions "`ASCII`" or "`text`". In addition to the requirements given above, your program must reject invalid inputs, handle various error cases to prevent fatal errors, return proper error messages to the user.

4. Your Tasks

This section reminds what you need to accomplish in this homework. First, you are asked to write `mfind` in the C programming language. Your program may consist of one or multiple C source code files. You must include a build script (e.g, `Makefile`) and a `REAMDE` document that instructs how to build and run your program in your homework submission. Note that your program will be tested on `peace.handong.edu`, thus, you are recommended to check if your program works well with the server.

You must submit a video presentation which must cover the following discussions:

- Present the overall structure and workflow of your program.
- Detail how you design and implement a linked list-based waiting queue, especially, on your uses of synchronization operations.
- Explain all cases of synchronization operations in your program.
- Demonstrate your program with test scenarios to evidence that your program satisfies all requirements. As a part of the demo, show that multiple worker processes are running concurrently.

Your presentation must not exceed 7 minutes. You must use English in the slide and narration. For supplementation of your presentation, you may put subtitle in Korean if you want. Evaluation will be primary based on your presentation, thus, carefully address all key points of your results in the presentation video. And be careful not to overstate your results. It will be treated as a fraudulent attempt if your statement and result are not consistent.

5. Submission

Submit an archive (e.g., as a `tar` or `zip` file) of all source code and results at a submission repository entitled with "Homework 5" in Hisnet by **11:00 PM, 23 June (Wed)**. The archive must contain (1) README file with a link to your presentation video (a link to YouTube, Vimeo, or Google drive), (2) all source code files and build scripts needed for reproducing what you reported in the video.