Operating Systems. Homework #4.

Due January 3, 2019, 23:59.

In this assignment, you will use tools you've learned to create a program that searches for files in the file system. The program will parallelize its work to enhance performance.

To complete this assignment, learn and use the following (not necessarily all): ***pthread_create, pthread_join, pthread_cond_wait, pthread_cond_signal, pthread_cond_broadcast, pthread_exit, pthread_cancel***, **pthread_mutex_lock/unlock, readdir, stat**

Cheating Policy

Part of the course requirements is to program assignments by yourself. You can discuss concepts and ideas with others, but looking at other people's code, looking at code from previous semesters, sharing your code with others, coding together, etc. are all not allowed. Students caught violating the requirement to program individually will receive a "250" grade in the course and will have to repeat it. *Even if you're under pressure, ask for help or even don't submit, rather than cheat and risk having to repeat the course.*

If you put your work in your TAU home directory, set file and directory permissions to be accessible only by you.

Late Submission Policy

You have 5 grace days throughout the semester, you can use all of them, some of them or none of them for this exercise. A 10 point deduction will be applied for every late day past the grace days.

Submission Guidelines

Submit a single file named **distributed_search.c**. **Document it** properly – with a main comment at the beginning of the file, and an explanation for ***every non-trivial*** part of your code. Help the grader understand your solution and the flow of your code.

We check

1. Program creates the right number of threads and uses them all to parallelize the search.
2. C program compiles without warning with "`gcc -O3 -Wall -std=gnu99 -pthread`"

Assignment

Write a C program named `distributed_search.c`. The program gets three variables as command line arguments

Argument 1 – Search root directory (search for files within this directory and its subdirectories)
Argument 2 – Search term (search for file names that include the search term)
Argument 3 – Number of threads to be used for the search (assume a valid number greater than 0)

The program performs the following flow:

1. Create a queue that holds directories.
2. Put the search root directory (where to start the search) in the queue.
3. Create *n* threads (the number received in argument 3). Each thread removes directories from the queue and searches for file names with the specified term. The flow of a thread is described below.
4. When there are no more directories in the queue, and all threads are idle (not searching for content within a directory) the program should exit with exit code 0, and print to stdout how many matching files were found. (`printf("Done searching, found %d files", num_found)`)

Threads should perform the following flow:

1. Dequeue one directory from the queue. If the queue is empty, sleep until it becomes non-empty. **Do not** busy wait (wasting CPU cycles) until the directory to become non-empty.
2. Iterate through each file in that directory.
3. If the file name contains the search term (argument 2, **case sensitive**), print the full path of that file (including the file's name) to stdout (`printf("%s\n", full_path)`)
4. If the file is a directory, don't match its name to the search term. Instead, add that directory to the shared queue and wake up a thread to process it.
5. When done iterating through all files within the directory, repeat from 1.

Important: The thread flow above is high-level. It doesn't describe how to detect when all directories have been processed and the thread should exit. It also doesn't describe how to implement the signal handling requirements described later. These are problem that you need to solve.

Error handling

- If an error occurs in any of the threads, print a proper error message, exit that thread, but **don't exit the program**!
- The program should exit when one of the following occurs:
    - All threads have exited due to an error: exit the program with exit code 1.
    - There are no more directories in the queue and all current threads are done searching (idle). If no thread has exited due to an error during the execution, exit with exit code 0; otherwise, exit with code 1.
- Print error messages to *stderr*.

General requirements

- Your program should be thread safe. (For example, a directory should not be searched by different threads.)
- Make sure no deadlocks are possible in your program.
- Threads should not starve, meaning that if there are N directories and more than one thread, they should not be all processed by the same thread.
- Upon receiving a SIGINT, this is how the program should behave:
    - The program **should not terminate** immediately upon a SIGINT.
    - When SIGINT is received, the program should let each thread exit gracefully, and then exit with exit code 0 after printing to stdout how many matching files were found. (`printf("Search stopped, found %d files.", num_found)`)
    - For a thread to exit gracefully, read about and use *pthread_cancel* and use any additional system call needed for its use (e.g. *pthread_testcancel* to force cancellation as soon as possible when a request has registered).
- The number of matching files found printed when the program exits must be equal to the number of file path names printed by the program.

If you're unsure about the guidelines of the exercise – ask in the forums. **As always**, closely follow the forums and all questions & answers provided there. Explanations, guidelines and relaxations may be given there, and they are **mandatory**.