CSCI 4210 — Operating Systems
CSCI 6140 — Computer Operating Systems
Homework 3 (document version 1.0)
Multi-threading in C using Pthreads

Overview

- This homework is due by 11:59:59 PM on Thursday, November 3, 2016.
- This homework will count as 9% of your final course grade
- This homework is to be completed **individually**. Do not share your code with anyone else.
- You **must** use C and the Pthreads library for this homework assignment.
- Your program **must** successfully compile and run on Ubuntu v14.04.5 LTS or higher.
- Your program **must** successfully compile via `gcc` with absolutely no warning messages when the `-Wall` (i.e., warn all) compiler option is used. We will also use `-Werror`, which will treat all warnings as critical errors.

Homework Specifications

Based on the first homework, in this third homework, you will use C and the POSIX thread (Pthread) library to implement a single-process multi-threaded text manipulation program. The goal is to work with threads and synchronization, parallelizing your program to the extent possible.

Overall, your program will open and read a directory of text files, parse all words from all of these files, store each parsed word into a dynamically allocated array, then display words that contain a specific substring. You can use your code from the first homework, but note that each file must be read and processed by a separate child thread. Use the main thread to orchestrate everything. More specifically, the main thread reads the given directory, assigning threads to regular files that it encounters. The main thread also allocates the initial array of character pointers.

All specifications from the first homework apply, except for the following:

- Each child thread must open its assigned file.
- The directory to be read is specified as the first command-line argument.
- Instead of storing just words, you must also store which files the words are in. Therefore, you should probably use a `struct` for this. This memory **must** be dynamically allocated.
- Child threads are responsible for re-allocating memory, as necessary. More specifically, if a word is to be stored, but the dynamically allocated array is full, the child thread detecting this must call the `realloc()` function. Such operations **must** be synchronized.
- The main thread performs the substring matching once all child threads have terminated.
- The required output differs as shown on the next page.

Required Output

When you execute your program, each child thread must preface its output with its thread ID, which you can obtain via `pthread_self()`. Further, each child thread displays the word that it processed, as well as at which index it places the word. As an example, if you are given a directory called `somedirectory` that contains the `filename.txt` file from the first homework and the following `xyz.txt` file:

```
The quick brown chicken jumps.
```

Executing the code as follows will display the output shown below. Some of the lines below could be interleaved in different ways, including the order in which the index numbers are displayed.

```
bash$ ./a.out somedirectory hi
MAIN THREAD: Allocated initial array of 8 character pointers.
MAIN THREAD: Assigned "xyz.txt" to child thread 1234.
MAIN THREAD: Assigned "filename.txt" to child thread 5678.
THREAD 1234: Added "The" at index 0.
THREAD 1234: Added "quick" at index 1.
THREAD 1234: Added "brown" at index 2.
THREAD 1234: Added "chicken" at index 3.
THREAD 5678: Added "Once" at index 4.
THREAD 5678: Added "when" at index 5.
THREAD 5678: Added "a" at index 6.
THREAD 5678: Added "Lion" at index 7.
THREAD 1234: Re-allocated array of 16 character pointers.
THREAD 1234: Added "jumps" at index 8.
...
THREAD 5678: Added "running" at index 15.
THREAD 5678: Re-allocated array of 32 character pointers.
THREAD 5678: Added "up" at index 16.
...
THREAD 5678: Re-allocated array of 64 character pointers.
...
THREAD 5678: Re-allocated array of 128 character pointers.
...
THREAD 5678: Re-allocated array of 256 character pointers.
...
MAIN THREAD: All done (successfully read 189 words from 2 files).
MAIN THREAD: Words containing substring "hi" are:
MAIN THREAD: chicken (from "xyz.txt")
MAIN THREAD: him (from "filename.txt")
MAIN THREAD: this (from "filename.txt")
MAIN THREAD: his (from "filename.txt")
MAIN THREAD: him (from "filename.txt")
MAIN THREAD: his (from "filename.txt")
MAIN THREAD: him (from "filename.txt")
MAIN THREAD: this (from "filename.txt")
...
```

Synchronization

You **must** parallelize your program to the extent possible, which will require synchronization among all threads; however, do **not** just implement the simple approach of merely having all threads share a single "mutex" variable. Instead, implement the following:

- When a child thread adds a word, the only part that is synchronized is obtaining the designated index (i.e., such that no child threads aim to write to the same array slot). Allocating memory and writing to the designated array slot must be able to occur in parallel with other threads. In other words, multiple threads could be writing to slots of the array simultaneously.
- When memory for the array needs to be re-allocated (i.e., by doubling its size), only one thread can do this. All other threads must be barred from accessing the shared array while this re-allocation occurs.

Handling Errors

Your program must ensure that the correct number of command-line arguments are included. If not, display an error message and usage information to `stderr` as follows (and return `EXIT_FAILURE`):

```
ERROR: Invalid arguments
USAGE: ./a.out <directory> <substring>
```

If a system call fails, use `perror()` to display the appropriate error message, then exit the program by returning `EXIT_FAILURE`.

If a thread call fails, display the error to `stderr`, matching the code examples shown on our course website.

Be sure that all dynamically allocated memory is freed via `free()`.

Submission Instructions

To submit your assignment (and also perform final testing of your code), please use Submitty. The URL is on the course website.