Operating Systems. Homework #4

Submission – May 31, 23:55.

In this assignment, we create a program that simulates a RAID4/5 recovery process. Our program receives the name of an output file and a list of input files, and creates the output file by XORing all input files. Thus, byte *i* of the output file is equal to the XOR of byte *i* of all input files whose length is at least *i*.

In addition, we will split the work among several threads, creating a separate thread for each input file, and synchronize the progress of the threads.

We provide a high-level description of how your program should work and some constraints on the design. It is **your** task to design an implementation that meets these requirements; this is part of the challenge of the assignment.

Program Specification

The program receives 2 or more command-line arguments. The first argument is the name of the output file, and the rest of the arguments are the names of the input files.

The goal of the program is to create the output file, such that its contents is the XOR of the contents of all input files. The value of byte *i* in the output file is the XOR over all bytes *i* of all input files whose length is at least *i*.

The program reads the input files in chunks of **1024 KB (2^{20} bytes)** and also writes the output file in chunks of 1024 KB. We compute the *j*-th output chunk by reading the relevant input chunks, XORing their data, writing the result to the output file, and only then proceed to the next output chunk. The last chunk may be smaller than 1024 KB, in which case we write whatever size is left.

Input files are not guaranteed to be of the same length! If some input file is shorter, its contents does not participate in the XOR of bytes beyond its length – it might even participate partially in a chunk! The length of the output file should thus be equal to the largest input file.

The program creates one reader thread for each input file. No other thread may access that input file. However, all threads may write to the output file. The rule is that the last thread that XORs its data into an output chunk will also write that chunk's contents to the output file.

Notes:

- If the output file already exists, truncate it, i.e., leave no trace of the original file.
- It is guaranteed that at least 2 arguments are given. There is no maximum guarantee on the number of arguments.
- If only 2 arguments are given (i.e., a single input file), the program should proceed as usual, with all synchronization – a single thread is created which reads the single input file and writes the output. The resulting output file should be equal in contents to the input file.
- On any error, print an error message and exit with a non-zero exit code. There is no need to recover the output file, free memory, close file descriptors, terminate threads properly, etc.

Detailed Description

Main Thread

The main thread is responsible for initializing all variables (locks, condition variables, output chunk buffer, whatever you need) and creating the reader threads. It then waits for all reader threads to finish, and terminates.

Flow:

1. Print the following welcome message: "Hello, creating <output> from <num> input files", where *<output>* is the name of the output file and *<num>* is the number of input files.
2. Initialize all needed variables
3. Open the output file for writing with truncation (use O_TRUNC)
4. Create a reader thread for each input file
5. Wait for all reader threads to finish
6. Close output file
7. Print the following termination message: "Created <output> with size <size> bytes", where *<output>* is the name of the output file and *<size>* is the output file's size.
8. Exit with exit code 0

Reader Threads

Each reader thread is assigned an input file. The reader threads implement the algorithm described in the specification above. The algorithm consists of a sequence of steps, and each step consists of the following parts:

1. The reader threads read the next 1024 KB chunk from their input files. (Reader threads who have already reached EOF in previous steps don't participate in this step.)
    - Reading must be done using the `read()` system call, not with any other library function or system call.
2. Each thread XORs the data in the chunk that it just read into a shared buffer.
3. The last thread to XOR its data also writes the result to the output file, and the next step begins.
    - Writing must be done using the `write()` system call, not with any other library function or system call.

Make sure you follow the following requirements:

1. In each step, the reader threads should read their chunks in parallel.
2. No reader thread should start part 2 of step *i* before part 3 of step *i-1* is completed.
3. A reader thread that reaches the end of its input file in part 1 of step *i* should terminate after step *i* is finished.
4. Each chunk from each input file should be read *once*, the XOR into the shared buffer should be performed *once*, and each chunk is written to the output file *once* (in each step).

Synchronization

It is up to you to implement the synchronization between the threads. Any solution that meets the requirements detailed above is acceptable. The only constraint is that you may use **only** `pthread_mutex_t` and `pthread_cond_t` types for synchronization, and not any other `pthreads` synchronization type.

Testing

To test your work, we first recommend that you implement the program without threading.

Create a program which creates no threads, in which the main thread does all the work – it opens all input files, iterates over them by reading the relevant chunk from each, XORs the data and outputs it, repeatedly until the output file is ready and all chunks have been written.

Using this program, you can compare the (hopefully correct) results with multi-thread executions.

You do not need to submit this program.

Submission

1. Submit a single file: `hw4.c`. Submit the file in a ZIP file named `ex4_012345678.zip`, where 012345678 is your ID. ***Mac Users: don't submit hidden folders!***
2. Your code must compile without warning with the following compilation command:

```
gcc -std=c99 -O3 -Wall hw4.c -pthread
```