TDT4200-PS3b – OpenMP and Pthreads

Zawadi Berg Svela and Jacob Odgård Tørring

Deadline: Sun October 11, 2020 at 22:00 on BlackBoard

In this assignment, you will again parallelize the password cracker presented in PS1. This time you will program for shared-memory systems, using POSIX threads (Pthreads) and OpenMP. For a detailed description of the password cracker application, see the assignment text for PS1.

1 NOTE

PS3a will be a short Quiz on BB based in this week´s Kahoot!. Note also that all assignment have to be done individually. Copying or close collaborations with classmates is considered plagiarism and will be dealt with accordingly.

2 Provided Files

- Folder with code for the OpenMP part of the assignment, with TODO's.
- Folder with code for the pthreads part of the assignment,with TODO's.
- A set of shadow files containing passwords to be cracked.
- A set of dictionary files containing symbols (words or characters) to try while cracking.

3 Problem description

For an introduction to The POSIX threading interface (Pthreads) and OpenMP, see Pacheco Ch 4 and 5, as well as the lecture notes from September 28th. [1].

In both parts of this assignment, you should only edit the function **crack()** in the *main.c* file, in the sections marked "TODO", and leave the rest of the provided code alone.

[1] `https://people.eecs.berkeley.edu/~demmel/cs267_Spr15/Lectures/lecture06_sharedmem_jwd15_4pp.pdf`

Task 3a.1 Pthreads

There are five parts of this task:

1. Initialize an array of thread ID's of type **pthread_t** with size equal to **MAX_THREADS**.
2. For each thread ID, spawn a new thread. **pthread_create()** expects a pointer to an argument structure, which is already provided.
3. For each thread, join it. This ensures that the main thread does not continue until all threads have finished.
4. Make and run the code. Ensure that it correctly finds the passwords.
5. Experiment with different thread counts. Set the **MAX_THREADS** value in line 160 to different numbers and note the run-times.

Task 3a.2 OpenMP

There are five parts of this task:

1. Set the variable to the OpenMP maximum number of threads. This is the default number of threads OpenMP will run.
2. Change the directive to run the region in parallel rather than single threaded.
3. Line To ensure that each thread works in a different job, get the OpenMP thread index.
4. Make and run the code. Ensure that it correctly finds the passwords.
5. Use the OpenMP wall time function to measure the timing of the jobs. Note the run-time and the number of threads.

Task 3a.3 Comparisons

1. Provide a table with timing results of the two programs using the same amount of threads.
2. How long did your Pthreads solution take compared to OpenMP?