IC221 pthreads lab Name FRANCESCONI

Spring AY22

**Learning Objectives**:

- Understand how POSIX threads are created and terminated (rejoined)

**Estimated Completion Time**: 1.5 hours **Lab Total**: 100 points

**Submission**: This completed document to the online submission site. No need to submit the C files.

**Example Files**: counters.c crack.c

**Task 1**: counters.c

Review the contents of counters.c. Examine the pthreads code. Understand how threads are created and joined.

Compile and run counters.c

gcc -o counters counters.c -lpthread

./counters

Note how the lpthread flag tells the compiler to link against the pthread library. This is necessary because it is not a 'standard' library. If you include this flag at the beginning, as in:

gcc -lpthread -o counters counters.c

it will cause a compiler error, because the source/target arguments are expected first.

(5) Do the counter threads *execute* in strictly numerical order? Why or why not?

|  |
| --- |
| No they do not execute in strictly numerical order. This happens because all threads execute simultaneously unlike processes which must wait for each other. |

(5) If you run the program repeatedly, does the order of thread execution sometimes change? Why or why not?

|  |
| --- |
| The order of thread execution changes each time because the CPU will decide the order in which the threads finish. |

(10) If one of these threads happened to block, would the other threads block? Why or why not? If you're not sure, uncomment the "lucky number" portion of thread\_function to experiment.

|  |
| --- |
| The other threads are not blocked because they do not need to wait for the thread that is blocked because none of the resources are locked so they do not need to wait to access them. |

**Task 2**: crack.c

Complete the crack.c code

The crack.c code is a password cracking program. It uses a brute-force approach to try to find the password that matches a given MD5 hash. The MD5 hash, written in hex, is called a *digest*. These are unsalted hashes. To keep it simple, we'll only uses password characters between 'a' and 'z', and we will crack passwords only of length 5. **Note**: if running on WSL or a VM, ensure you have the ssl library: sudo apt install libssl-dev

Each of you will receive a unique batch of digests to crack, randomly selected based on your alpha code. All passwords are real user passwords, randomly chosen from a published batch of compromised passwords. Your digest file will have the form m999999.digest

In the program's main() function, it opens the file containing the digests and reads them into memory. Edit the appropriate line so the program opens *your* unique digest file, associated with your alpha code.

Compile the program as follows: gcc crack.c -o crack -lpthread -lcrypto. Notice the addition of both the pthreads library and the crypto library that we are linking against.

Run the program: ./crack

The program should take less than a minute to crack all the hashes, but way of brute force. It starts at 'aaaaa' and tries every possible password up to 'zzzzz'. This is not optimally efficient, but that's okay.

(10) Enter the output of the program, showing that it broke all the passwords in your batch:

|  |
| --- |
| Thread 1 Crack digest: db70804649dd13f6fdce2fc52dc411cf  Thread 2 Crack digest: 2642b9054daf77ae8647f14f58421f45  Thread 0 Crack digest: 8d48606bf9e1e8cb0dd9d6799189cc3c  Thread 3 Crack digest: 726a1e45735884cbf5a7b57a8f39c832  Thread 4 Crack digest: 7b8b41f15cbd2efcc21fe6adcb1e38ec  Thread 5 Crack digest: 60d3bbad4f50021cddeaed7902f1f796  Thread 6 Crack digest: 4d57c5e47b0c0a71a8c1b57cf1d9c8b1  Thread 8 Crack digest: eedea3117cabea784bdffee0420be4d0  Thread 10 Crack digest: 9aeb0e65a1aadcdb1f8508844f119f80  Thread 13 Crack digest: 335ffbcfac66fb164e9d5a54505cad8f  Thread 6 Match: betta 4d57c5e47b0c0a71a8c1b57cf1d9c8b1  Thread 5 Match: bolis 60d3bbad4f50021cddeaed7902f1f796  Thread 2 Match: flute 2642b9054daf77ae8647f14f58421f45  Thread 8 Match: hokey eedea3117cabea784bdffee0420be4d0  Thread 4 Match: krina 7b8b41f15cbd2efcc21fe6adcb1e38ec  Thread 0 Match: luger 8d48606bf9e1e8cb0dd9d6799189cc3c  Thread 10 Match: kobus 9aeb0e65a1aadcdb1f8508844f119f80  Thread 13 Match: oasis 335ffbcfac66fb164e9d5a54505cad8f  Thread 1 Match: tikka db70804649dd13f6fdce2fc52dc411cf  Thread 3 Match: uscio 726a1e45735884cbf5a7b57a8f39c832 |

Alpha Code: 251806

(10) Use the time command to time the execution, as follows: time ./crack

How much does the OS report as the *real* time needed to break your digests? Ignore the *user* and *sys* times.

|  |
| --- |
| 12.567s |

Multithreading. Modify the code as follows:

- Add a loop in main() to create NTHREADS pthreads. Use the array statics[i] as an argument, so they will get assigned a meaningful thread number.

- Add another loop in main() that will join them all when complete.

- Comment out the last two lines of main(), which invoke the cracking function using a single thread.

(20) Compile and re-run your code using NTHREADS values of 2, 4, and 8, respectively, and complete the table below. If running a Linux VM, I encourage you to assign the VM at least 4 CPUs for this exercise. If the program hangs, just break out with CTRL-C and note that.

|  |  |
| --- | --- |
| Number of Threads | "Real" time to crack all digests (seconds) |
| 1 | 51.188s |
| 2 | 27.640s |
| 4 | 16.958s |
| 8 | 12.999s |
| 16 | 12.687s |

(10) Run the command lscpu from the command shell to determine the number of cores on the Linux machine you are using. Look at the entry "On-line CPU(s) list" for a range, or multiply "Cores per socket" times "Threads per core". Enter the number you calculated:

|  |
| --- |
| 12 |

(10) What performance trend did you observe, after implementing multi-threading?

|  |
| --- |
| More threads makes the program run faster |

(20) Was there a certain number of threads above which no notable performance change was observed? If so, what was the number, and why do you suppose that is?

|  |
| --- |
| There was no noticeable performance change between 8 and 16 threads. I believe this is because there are 12 cores so at most 12 threads can be run at the same time so having more than 12 threads will not improve performance without more cores. |